RELIABLE HEALTH MONITORING: A COMMERCIAL OFF-THE-SHELF AND A FIELD PROGRAMMABLE HARDWARE APPROACH

BY

AJAY MATHEWS CHERIYAN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Advisers:

Professor Ravishankar K. Iyer
Research Professor Zbigniew T. Kalbarczyk

ABSTRACT

With the tremendous advancements in low cost, power-efficient hardware and the recent interest in biomedical embedded systems, numerous traditional biomedical systems can be replaced with smaller and faster embedded systems that perform real-time analysis to provide bio-feedback to the users. This thesis takes a look at two hardware implementations – one using commercial off-the-shelf (COTS) components and the other using field programmable logic.

The focus of the design was to ensure a portable, inexpensive, power-efficient and robust device that could perform analysis of physiological signals, which would in turn help alert the user in the event of an abnormality. The COTS hardware implementation provided the framework using a microcontroller as the processing element for a reliable health monitoring device with a seizure detection directly embedded in it.

The field programmable gate array (FPGA) platform based implementation was proposed and simulated to overcome the two disadvantages of the COTS approach – the inability to support customization of the device to suit the end-user's monitoring requirements and complex detection schemes requiring significant processing capability. The FPGA platform was simulated first as a standalone module and later as part of an SoC design. The novel algorithm included a feature extraction phase and a machine learning based seizure detection phase. Simulation based testing of the device showed a detection accuracy of 99.2 %.

ACKNOWLEDGMENTS

This thesis was built upon the foundation laid by many people, without whom I would not have been successful in my efforts. Foremost, I thank the Lord God Almighty for blessing me with the wisdom and providing me with the guidance needed to work on this project and make this submission. I am grateful for the support and guidance offered by my advisers, Ravishankar K. Iyer and Zbigniew T. Kalbarczyk. This effort would not have been possible without the advice and guidance of Ken Watkin, who was instrumental in the development of this research project. I sincerely appreciate the efforts of my group members, Albert O. Jarvi and Mushfiq Salaheen, who played pivotal roles during various stages of this project's development. Additionally, this work draws upon the financial support of the TCIP project and US Army grant. I thank my family and friends for supporting me during this process.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This thesis presents two approaches to designing an embedded system that is capable of robustly detecting abnormalities in the health of an individual. Specifically, these systems have been adapted to implement algorithms that are capable of detecting seizures. To this end, the initial design focused on using commercial off-the-shelf (COTS) components to implement a hardware platform that is used to detect seizures. An improved design focuses on using field programmable gate arrays (FPGAs) to implement a machine learning based detection algorithm that adds support for a user programmable interface, and is implemented as a standalone module and as part of the Leon3 soft-core processor.

1.1 Healthcare Monitoring

Many devices in the medical field continue to require significant processing power because of the vast amounts of data processing involved in acquiring physiological signals. Only recently has it become feasible to utilize this increased processing power for embedded and cyberphysical systems. Cyberphysical systems differ from traditional embedded systems in that the focus tends to be on the interaction between the computational and physical elements. A number of similar applications have come up in recent times with the prominent one among them being body sensor networks [1]. These networks utilize modified wireless sensor network platforms for biomedical applications.

An example of such a system is the SMART Attire or SATIRE [2], which provides wearable computing to monitor the activities of a person.

New devices have also emerged on the market to make biomedical monitoring possible in an automated and reliable manner. The possibility of automated diagnosis of abnormalities by using low power devices and the emergence of prototype devices in this nascent stage is indeed the impetus for this thesis.

1.2 Contributions of this Thesis

This thesis presents the design of an embedded system that is capable of collecting data from physiological sensors in a synchronous manner and analyzing the data in real time to detect abnormalities in the waveforms and warn the user in due time. Two hardware implementation platforms were chosen to demonstrate the feasibility of the design – one using COTS components which include microcontroller and inexpensive, non-invasive sensors, and the other using FPGA logic. The implementations focus on detecting abnormalities in the electrical activity of the brain known generally as seizures. The FPGA scheme has a novel seizure detection algorithm with a very high accuracy of 99.2 %. The FPGA scheme is implemented in two ways- the first is suitable for a custom ASIC implementation while the second introduces a flexible hardware-software implementation suitable for SoC designs. The thesis analyzes the features of all implementations and the advantages of each method along with the algorithm that is used to detect abnormalities in the signals collected.

1.3 Organization

The rest of this thesis is organized as follows. Chapter 2 explains related work, its differences from this work and provides some background information in brain activity monitoring. Chapter 3 explains the COTS implementation of the embedded system. Chapter 4 deals with the analysis of the seizure detection algorithm used in the COTS approach. Chapter 5 deals with the FPGA implementation of the novel seizure detection scheme. Chapter 6 presents the framework for the Reliable Biomedical Processing Engine along with its implementation. Chapter 7 presents the results of the proposed detection algorithm used in the FPGA seizure detection system and Chapter 8 presents the conclusions drawn and some direction for future research.

CHAPTER 2

RELATED WORK AND BACKGROUND

The hardware design and detection schemes presented in this thesis relate to detection of seizures. Hence, the following sections present information related to the detection and analysis of seizures. Section 2.1 deals with related work and Section 2.2 provides some background in brain activity monitoring.

2.1 Related Work

Hively et al. [3], under the CRADA hardware and software setup, utilize non-linear techniques in EEG forewarning equipment. This method works by utilizing a three-dimensional phase space representation of the collected data. Seizure occurrence is predicted from the dissimilarities between the distribution functions for the non-seizure and seizure waveforms. However, this device is exclusively limited to predicting seizures. Environmental conditions under which the patient is operating are not taken into account. These conditions could impact the decision-making process about whether the seizure could eventually manifest as something more critical or fatal. These two constraints limit the application space of the device. Verma and colleagues [4], [5], [6] focus on small-footprint system-on-chip (SoC) solutions for continuous patient EEG monitoring and seizure detection. The SoC implementation focuses only on real-time data collection.

The system designs presented here differ from the above mentioned works in their ability to carry out real-time signal collection and processing to alert the user in case of an abnormality detection. The COTS approach shows the ability of simple and inexpensive devices to provide the required functionalities with less flexibility, while the FPGA approach provides the advantages of the COTS approach along with a certain level of flexibility in that the user can configure the device to suit his detection requirements. Additionally, the FPGA scheme will be able to support fast, real-time, multi-dimensional signal analysis using statistical techniques and signal processing algorithms, which is extremely useful in biomedical patient monitoring devices.

Considerable work has been done in the area of embedded devices for collecting EEG data with reduced artifacts [7], [8]. Such devices or solutions focus only on the data collection aspect of the EEG data with almost no processing carried out in real time. [9] describes a real-time EEG processing device based on TI's signal processing platform which decomposes the collected EEG data into various frequency bands for visualization purposes. However, the power consumption of such a device is quite high and prevents its use in a power-constrained embedded environment.

While there exists hardware to monitor biomedical signals, very few devices exist which have an abnormality detection algorithm embedded in hardware, in addition to flexibility with regard to the device's capabilities; i.e., they are focused exclusively on monitoring one or two physiological signals. Aurelia Microelettronica has been developing integrated circuits for monitoring ECG and other vital parameters since 2001. CARDIC [10] is a device developed by the same company which has a low power multi-

sensor front-end acquisition system developed mainly for the acquisition of electrocardiographic signals. [11] introduces a portable intelligent ECG monitor with an algorithm implemented in software utilizing R-peak and QRS complex detection in ECG signals to identify abnormal heart beats. While it is capable of monitoring ECG and diagnosing abnormalities, the application space is limited to monitoring only ECG signals. [12] addresses wireless biomedical monitoring devices implemented as ASICs which can be used to monitor EEG, ECG, EMG, and EOG, and to transmit them wirelessly. The device is only capable of monitoring signals, with no processing being done on-chip; thus, it is significantly different from the reliable monitoring and flexible hardware based implementation described here.

2.2 Background

Monitoring brain activity has been useful in detecting and explaining brain injuries and disorders in individuals. Two commonly used technologies are pulse oximetry and electroencephalography [13], [14].

2.2.1 Electroencephalogram (EEG)

For many years EEG has been used extensively to monitor brain activity. EEG is the recording of electrical activity along the scalp of a person produced by firing of neurons within the brain. In clinical contexts, EEG is recorded over a short period of time. The main diagnostic application of EEG is in the case of epilepsy with the major abnormality being referred to as an epileptic seizure. Any abnormal activity in the EEG is generally

classified as a seizure. If the seizure is related to epilepsy, it is characterized as an epileptic seizure; otherwise, it is non-epileptic. Seizures are classified as

a) Partial seizures / focal seizures
b) Complex partial seizures / psychomotor seizures
c) Generalized seizures

More details on the various types of seizures can be found in [15], [16]. The seizure data that was used to verify the functionality of the algorithm in the COTS implementation was obtained from a study that involved patients suffering from epilepsy. Additional details are included as part of Chapter 5**.**

One difficulty with EEG is that only trained clinicians are able to interpret EEG waveforms and identify abnormalities. Quite recently, however, studies have focused on utilizing digitized EEG data to extract useful information. This is called quantitative EEG (QEEG). QEEG helps to transform the waveforms into frequency bands using FFT or other filtering techniques. Once a pattern in such a scheme is associated with an abnormality, there exists a mechanism for automated decision making. The rhythmic activity in EEG is generally classified into the following bands:

a) **Delta wave -** It includes all frequencies up to 4 Hz. Delta wave activity is quite common in the waking EEG of infants while abnormal in healthy adults.
b) **Theta wave** – It includes the frequencies in the 4-7 Hz range. Excessive theta wave activity is generally perceived as abnormal in active adults.
c) **Alpha wave** – It includes the frequencies in the 8 – 12 Hz band. Alpha wave amplitude is typically higher on the dominant side of the individual and is brought

about by relaxation. It is generally noted to attenuate with mental exertion or eye opening.

d) **Beta wave** – All waves in the 13 – 24 Hz are classified as beta waves. It is seen usually on both sides of the individual in symmetrical distribution. Beta activity is generally attenuated during active movements.

As part of the detection scheme embedded in the COTS implementation, the focus was on the alpha and theta bands. In the seizure data that was analyzed, abnormal activity was detected primarily in the alpha and theta bands. A more detailed explanation of this is included in Section 4.1 and 4.2.

2.2.2 Pulse Oximetry

Oxygen saturation or SpO_2 can be measured using pulse oximetry. One benefit of pulse oximetry is that it is non-invasive. Studies have shown that oxygen saturation is an efficient indicator of neurological outcome in patients with traumatic brain injury. In normal adults, oxygen saturation remains close to 100%. Cognitive ability and normal brain functions begin to be affected when oxygen saturation is below 90%. Very poor neurological outcome is expected when oxygen saturation is below 70% [17], [18]. In some patients with epileptic seizures, oxygen saturation levels have been shown to drop below 90%, while some drop below 70%, soon after seizure onset [19], [20], [21].

CHAPTER 3

COTS SEIZURE DETECTION SYSTEM DESCRIPTION

This chapter describes the hardware implementation of the seizure detection system using COTS components. Non-invasive sensors for monitoring and collecting physiological signals are combined with a microcontroller to realize the system.

3.1 System Description

In this section, the various modules which form the system are discussed in detail. The various modules are accelerometers, oxygen saturation and heart rate monitor, EEG electrodes, a microcontroller and a wireless interface.

3.1.1 Accelerometers

Accelerometers help capture the acceleration of an object in terms of an analog voltage which is directly proportional to the dynamic acceleration experienced by an object. In other words, a device at rest will indicate a value of acceleration equal to 9.8 m/s^2 or 1 g along the vertical axis. However, in most cases, the value of acceleration would be non-zero along the other axes, which makes it necessary to calculate the baseline values of acceleration for a body at rest. The accelerometers are part of this design for its use in applications where the risk of injury is related with an action that results in quick movement of the person or object. The accelerometers used in this system are MEMSIC's heat transfer based micrometer sized devices which offered significantly

improved performance over traditional proof mass based systems and have a shock rating of over 50,000 g. The prototype uses models of the device with a higher sensitivity of 500 mV/g and range of +/-1.7 g at 3 V. This is to obtain a measurable voltage with reasonable movement under the lab's testing conditions. Accelerometers with lower sensitivity can be used when the total acceleration to be expected is higher. Tri-axial accelerometers are used in this prototype.

3.1.2 Oxygen saturation and heart rate monitor

Oxygen saturation measures the percentage of hemoglobin binding sites in the bloodstream occupied by oxygen. The device used to perform the calculation is called a pulse oximeter. It relies on the light absorption characteristics of saturated hemoglobin to determine the percentage of oxygen contained. The typical method of measurement consists of using a sensor containing red and infra-red light-emitting diodes. The diodes are placed in contact with the skin along with photodiodes to determine the amount of light from the two sources which are absorbed and reflected. This data is used to compute the oxygen saturation. As mentioned earlier, oxygen saturation under normal conditions is close to 100%.

The wave patterns, picked up by the photodiodes, display an ac value with a dc component. As can be intuitively understood, since we are computing the blood oxygen saturation levels, the frequency of the ac component gives us the heart rate of the individual. Hence, most pulse oximeters are capable of computing the heart rate. Due to the availability of commercial devices which are capable of computing these values

accurately with minimal artifacts, a commercial product was used for our purpose. The OEM III module from NONIN is used in this study.

The PureSAT signal processing technique employed in the module is ideal for use in motion and low perfusion environments. This approach provides more reliable readings over simple microcontroller based pulse oximetry solutions. The NONIN sensor provides a 4 beat average heart rate and 4 beat average SpO_2 values.

3.1.3 EEG electrodes

EEG refers to the measurement of the electrical activity of the brain and is recorded by placing multiple electrodes on the scalp. Electrode locations and names are specified by the International "10-20" system ensuring consistency in the naming convention. In most clinical applications, 19 electrodes along with two reference electrodes are used. Most of the EEG data used in this study is obtained from patients suffering from seizures. With regard to seizures, an important term is the *ictal* period. Ictal period is the duration of the actual seizure and the EEG reading during this period is the ictal EEG. The times shortly before and after the seizure are called *pre-ictal* and *post-ictal* respectively. The time between two seizures is the *inter-ictal* period and for epileptic patients, most of the EEG readings correspond to this period.

A significant amount of seizure activity is observed in the frontal region of the brain (i.e. in the electrodes placed on the forehead) and hence, most of the study and detection schemes used in this research focus on Fp1 and Fp2 electrodes. The waveforms observed on the two electrodes are quite similar. This can be expected because of the symmetrical

placing of the two electrodes on the forehead. Hence for our analysis, we focus on the signal from the Fp1 electrode. One recent study of EEG characteristics related to depressive disorder conducted with data collected during the ictal period of the seizure observed maximum change on the Fp1 electrode [22].

To develop and test the prototype, we utilize EEG signals generated by an EEG simulator (Grass Technologies, Model EEGSIM). The simulator has a patient's ictal EEG stored on a PROM (programmable read only memory) which is replayed every 60 s. Several models of the EEG simulator corresponding to different types of EEG waveforms are available.

3.1.4 Microcontroller

The central processing element of the device is a microcontroller which is capable of collecting the data from the different sensors and processing them. In addition to the processing power, limiting the power consumption was an important constraint. The MSP430FG4618 from the MSP430 line of microcontrollers was chosen for this purpose. The microcontroller has 116 kB of flash memory and 8 kB of RAM. The important peripherals included in the microcontroller are the analog-to-digital converter (ADC), the serial communication interface and the serial peripheral interface (SPI). It also features the support for an LCD module, which can be used to provide visual feedback to the user should the application demand it. A development board with the microcontroller was used for the prototype development.

3.1.5 Wireless interface

The device has been designed keeping in mind the desire to communicate the results of the monitoring procedure to a base station which is capable of logging the results periodically or to communicate the result to another piece of equipment. For this purpose, a wireless communication interface was also set up. The Chipcon CC2500 module from Texas Instruments is used for this purpose. It is a low power and low cost transceiver used for wireless communication in the 2.4 GHz ISM band. It is capable of transmitting in packets of 64 bytes along with support for a low power sleep mode operation.

3.2 System Operation

The signals from the sensors, namely the accelerometers, pulse oximetry module and EEG electrodes are first processed by the ADC before being collected by the microcontroller. The signals from the accelerometers can be low pass filtered if desired, but the dc component of the waveform should not be eliminated to obtain a higher analog voltage of acceleration. Low analog voltages can suffer attenuation resulting in faulty decision making. Two dual-axis accelerometers or a single tri-axial accelerometer can be used to obtain acceleration along the X, Y and Z directions. The microcontroller is not capable of sampling negative voltages and so, to facilitate that, the EEG signal is level-shifted by a dc value before being fed into the amplifier. Since the EEG signal levels are typically in the range of 5-50 μV, a high impedance biomedical signal amplifier is used to amplify the signal by a value of the order of a few thousands. The amplifier used supports variable gains from values of 1700 to 5000. A value of 2000 was used for our

experiments. The amplified output is fed into the microcontroller. The level-shifting voltage value used is also sampled by the microcontroller. The pulse oximetry module is interfaced with the microcontroller through the serial port and the oxygen saturation and heart rate values are obtained in digital format, so no processing is required.

The frequency at which these signals are sampled depends on the frequencies of interest. The frequencies of the EEG signals and accelerometer waveforms influence the minimum sampling rate. Since the accelerometer readings are instantaneous measurements of acceleration and since no frequency band decomposition is carried out, we will focus on the frequencies of interest in the EEG signals which are in the 1 – 70 Hz range. However, since the metrics we utilize rely only on information in the alpha and theta bands, the maximum frequency in the bands of interest is 12 Hz and so the sampling frequency should be greater than 24 Hz. A value of 64 Hz is chosen. Sampling within the microcontroller can be carried out by using timers to trigger the conversion process within the ADC or by setting the ADC to continuously sample the data from the sensors and then using timers to read the digital values at the sampling frequency. The former approach is used as it is more consistent with the idea of sampling. Two timers are used to generate a 64 Hz waveform which is used to enable conversions within the ADC module.

Since we need to extract information contained within the various frequency bands of the EEG waveforms, we use 21-point symmetric finite impulse response (FIR) filters. The coefficients for the filters are pre-computed using MATLAB and saved in the microcontroller. A 21-point value was chosen as a tradeoff between precision and

computational space. A higher precision filter could have been used at the expense of higher storage space requirements (each coefficient is a floating point value) without considerable improvements in the quality of the filtered output. Another important consideration is the epoch or time window over which the calculations are made. While it is desirable to make decisions over a relatively long period of time, the memory space requirements impose a burden because of the limited memory space within the microcontroller. A value of 12 s was chosen to be length of the time window. While the measurements made are dependent on the length of the time interval, eventually a comparison is carried out with reference to a baseline value computed over the same time interval for every individual. This offsets the impact of the length of the selected time window. All the metrics are computed over the duration of this 12 s window.

The entire system operates as shown in Figure 1. Once the module is switched on, it computes the baseline/at rest values for metrics pertaining to each of the sensor values. Then the module goes into a monitoring mode, which can be continuous or event-triggered. In the continuous monitoring scheme, data from the sensors are collected and metrics are computed on a continuous basis. In the event-triggered monitoring scheme, an external event is used to trigger the monitoring process. One example of such an event could be an accident resulting in sudden acceleration of the individual which would show up as a spike in the accelerometer reading. This spike can then be used to shift the module in the continuous monitoring mode. Such a scheme can be used in extremely power-constrained environments. Once the data is obtained, the results of the computation process are then wirelessly transmitted to the base station which is another

MSP430 node connected via the serial port to the PC. A Perl program reads the data from the serial port and logs the data into a MySQL database.

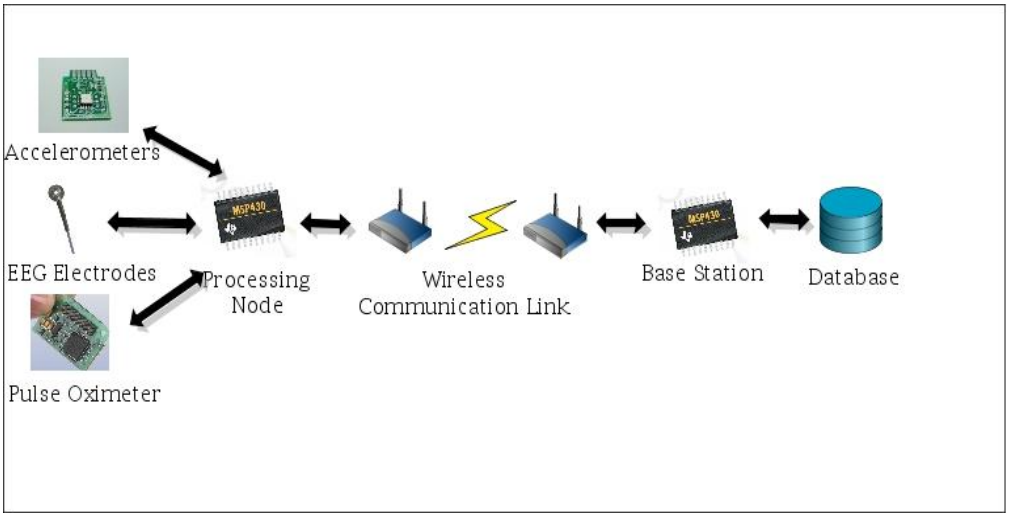

Figure 1: Working of the COTS Health Monitoring Implementation

CHAPTER 4

COTS SEIZURE DETECTION SYSTEM ANALYSIS

This chapter deals with the analysis of the seizure detection scheme that is embedded within the microcontroller used in the COTS system. Section 4.1 explains the detection algorithm and metrics used in the scheme and Section 4.2 validates the detection scheme using epileptic EEG data recorded in a database in Germany.

4.1 Detection Algorithm

As mentioned previously, the EEG data on which analysis is carried out was from a person who suffered a seizure. One minute of ictal data is available using the simulator. The primary indicator of a seizure is an increase in the amplitude of the low frequency theta waves with a simultaneous decrease in the amplitude of the higher frequency alpha waves. Since power is directly proportional to the square of the amplitude, the power contained in each frequency band can be computed over the duration of the data collection period (i.e., 12 s in our study). The metric that is used to detect abnormalities in the EEG waveform is therefore the *alpha-theta* ratio [23], [24].

The alpha-theta ratio can be defined as the ratio of the power in alpha frequency band to the power in theta frequency band. When a seizure is observed, the alpha-theta ratio for that individual should be lower than the base/normal value established for that person. Figure 2 shows the alpha wave during the 12 s pre-ictal and ictal periods and Figure 3 shows the theta wave during the same periods. Although the ictal period shows a minor

increase in the alpha wave amplitude, the theta wave amplitude increases almost threefold, resulting in an overall decrease in the alpha-theta ratio.

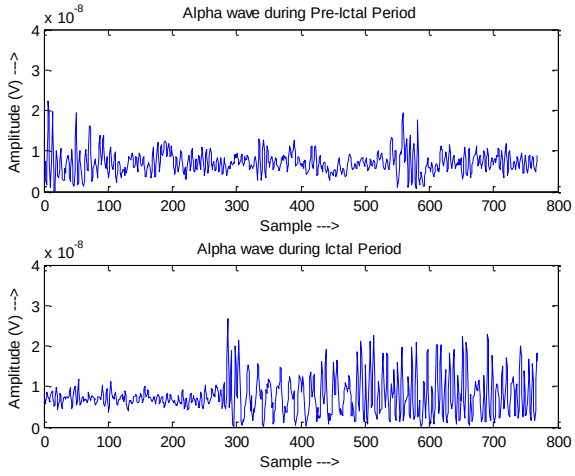

Figure 2: Alpha Wave during Pre-ictal and Ictal Periods

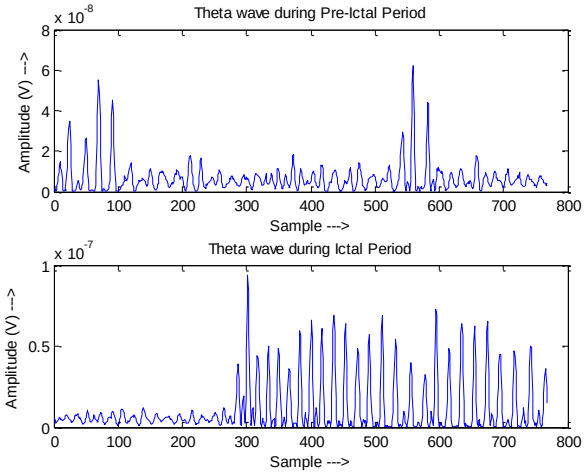

Figure 3: Theta Wave during Pre-ictal and Ictal Periods

Table 1 highlights the value of alpha-theta ratios computed from the one minute of EEG data obtained from the simulator. The lowest value in Table 1 is obtained when the continuous sequence of spikes are observed in the ictal waveform. As mentioned earlier,

a poor neurological outcome is expected when oxygen saturation levels drop below 70%. Recent studies have also shown oxygen desaturation to be correlated with the occurrence of the seizure. Hence, a simple indicator of abnormal oxygen saturation levels would be a drop below 70%. Another indicator of abnormal health conditions would be a very low heart rate. If the heart rate drops to less than 75 percent of the base reading, it is inferred to be abnormal.

Table 1: Alpha-Theta Ratios of EEG Simulator Data

Reading	Alpha-Theta Ratio
1.	0.3477
2.	0.3464
3.	0.4456
4.	0.5127
5.	0.6040

The metrics for each of the sensor readings are summarized in Table 2. All of the individual metrics take a value of 1 under abnormal conditions and 0 under normal conditions as outlined in Table 2. These individual metrics can be combined linearly into an overall Criticality Factor which indicates the level of critical injury suffered by the subjects.

Table 2: Metrics used for each Sensor

Property being measured	Metric Name	Normal Value (with respect to baseline readings)	Abnormal Value (with respect to baseline reading)
EEG	Critical EEG	Above 50%	Less than 50%
Oxygen Saturation	Critical Oxygen	Above 70%	Less than 70%
Heart Rate	Critical Heart Rate	Above 75%	Less than 75%

Criticality Factor = Critical EEG + Critical Oxygen + Critical Heart Rate

Based on this formula, the value of Criticality Factor can assume 4 values from 0 – 3 and the conclusions drawn from each of those values are shown in Table 3. An indicator light, based on the Criticality Factor, is flashed for quick reference. A green LED is flashed to indicate normal conditions, a yellow LED is flashed for a possible abnormality, and a red LED is flashed for critical injury.

Table 3: Inference from Value of Criticality Factor

Value for Criticality Factor	Inference
0	Individual is healthy
1	Abnormality only if Critical Oxygen is one and seizure if Critical EEG is one
2	Possible injury/seizure
3	Seek further medical care

4.2 Validation of Detection Algorithm

In this section, we validate the use of the alpha-theta ratio metric we discussed in the previous section for EEG seizure detection. The waveforms plotted in Section 5.1 and the values for the alpha-theta ration indicated in Table 2 are based on the EEG data obtained from the EEG simulator. While this is actual ictal EEG, values obtained on the basis of a single test may not be conclusive enough to validate the use of the proposed metric for the seizure detection.

Since this study did not involve trials on human subjects, we perform the same processing on actual EEG data, collected from patients suffering from epileptic seizures stored in the EEG database at Albert-Ludwigs Universitat Freiburg, Germany. The database contains invasive EEG recordings of patients suffering from medically intractable focal epilepsy. The datasets are recorded during an invasive pre-surgical epilepsy monitoring comprising 24 hours of ictal and inter-ictal EEG recordings. Experienced epileptologists visually inspected the intracranial recordings to determine the ictal periods based on identification of typical seizure patterns preceding clinically manifest seizures. The electrode positions are different for different people. Table 4 shows the values of the alpha-theta ratio calculated during the ictal periods and the value calculated during the inter-ictal period. The value obtained during the inter-ictal period is taken as the baseline value. Since the data is available in digital format, a MATLAB program performing the same processing as the microcontroller is used to obtain the values shown in Table 4. Figures 4 and 5 show the seizure patterns obtained for two patients using data from the study.

Table 4: Alpha Theta Ratios for Patients Suffering Epileptic Seizures

Patient Information	Seizure Number	Baseline Alpha-Theta Value	Ictal Alpha-Theta Value
15 year old female	1	0.7804	0.4575
	2	0.7804	0.54
	3	0.7804	0.76
14 year old male	1	0.9296	0.2560
	2	0.9296	0.1828
32 year old female	1	1.3943	0.2786
	2	1.3943	1.0228
	3	1.3943	0.9208

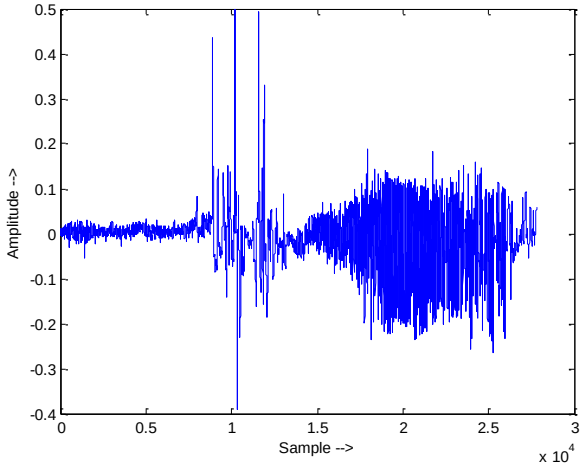

Figure 4: Seizure Pattern 1 - Alpha-Theta Ratio = 0.2560

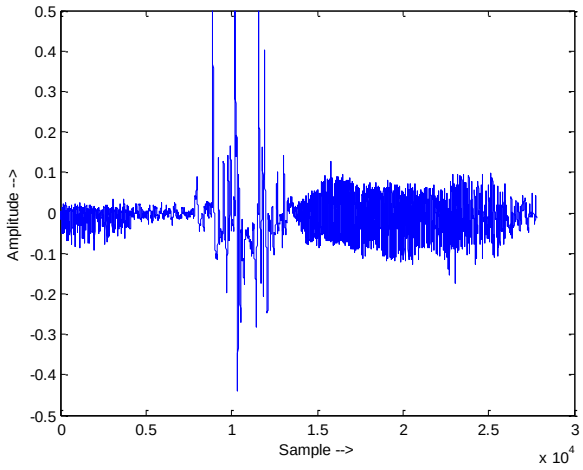

Figure 5: Seizure Pattern 2 - Alpha-Theta Ratio = 0.4575

As can be seen in Table 4, there is a significant decrease in the alpha-theta ratio in almost all of the readings analyzed. Seizure 2 of the 14-year-old patient showed the maximum change with an ictal alpha-theta value of 0.1828 while the baseline value was 0.9296. However in some of the readings a value close to baseline value is seen, although it is smaller than the baseline value. For example, seizure 3 gave a value of 0.76 during the ictal period for the 15-year-old female patient, which is close to the baseline value of 0.7804. There could be several reasons for this behavior which cannot be inferred without obtaining additional information about the other conditions of the patients. This validates the use of the metric for seizure onset detection. Figures 4 and 5 show two ictal seizure patterns that were observed along with the alpha-theta value obtained. The ictal period, similar to the one obtained from the simulator (used in our measurements) shows an increase in the amplitude (or spikes) and a decrease in the alpha-theta value over the baseline values.

4.3 Possible Downsides to the Detection Algorithm

The Freiburg seizure database had data from patients suffering epileptic seizures but portions of the datasets were identified as seizures in general with no mention to the type of the seizure. This could have a serious impact on the detection algorithm. This detection algorithm is based on an increase in theta wave amplitude after the onset of the seizure. This means that the EEG wave is gradually slowing itself down into the theta wave band. Also, the data collected in Table 4 were obtained from the frontal electrodes. This type of behavior is quite common in grand mal or tonic-clonic seizures. It is one of the most common forms of epileptic seizures. From past studies and literature surveys, it has been noted that these seizures exhibit a spiky pattern in the theta wave range and in the first half of the alpha wave range. It is also found to be more prominent in the frontal electrodes. This type of seizure has been noted to affect young adults more than any other seizure. This possibly explains the significantly better detection in the data obtained from the young teenagers in Table 4.

CHAPTER 5

FPGA-BASED SEIZURE DETECTION SYSTEM

In this chapter, we take a look at the design and architecture of a system which is capable of performing the same task as the COTS seizure detection system but with added advantages at the implementation level and in the quality of the detection algorithm used. The improved design uses FPGA logic to achieve the same goal. The disadvantages of the COTS system and the reasons for using FPGA logic are explained in Sections 5.1 and 5.2.

5.1 Disadvantages of the COTS Implementation

The major advantage of the COTS implementation was the ability of the system to be integrated using off-the-shelf components which were inexpensive and non-invasive. While that still remains an advantage, especially in terms of the sensors used, the major bottleneck with regard to the robustness of the system proved to be the microcontroller's processing power. The MSP430 is capable of running only at a much lower frequency than that of a more powerful processor. Even though the MSP430 has an integrated hardware multiplier, it is designed for integer arithmetic and floating point arithmetic is not handled well by most microcontrollers, including the MSP430 family. Also, a significantly more robust detection algorithm would require considerable floating point arithmetic, limiting the detection accuracy and speed of execution if implemented on a microcontroller. Also, the same platform would be able to support monitoring of multiple

physiological signals only by statically modifying the embedded code. The user has no control over the monitoring process.

5.2 Advantages of the FPGA Implementation

The FPGA implementation retains almost all, if not all, of the advantages that the COTS implementation provides in addition to providing flexibility, superior performance and a faster and more accurate detection scheme directly implemented as hardware logic. The same platform can be modified to support multiple detection algorithms, one each for every different symptom that is being analyzed. It can be used to provide additional features to improve the system flexibility. For example, each module can be either separately enabled or disabled by means of "toggle" instruction, or if not needed entirely, the chip can be synthesized without the unwanted module.

5.3 Original Detection Algorithm - Software

While simple, the detection algorithm used in the COTS implementation, as noted earlier, was able to accurately detect seizures that were similar in symptoms to the ones observed in the case of grand mal seizures. Being able to accurately identify all the different types of seizures would certainly be more advantageous. Hence, a detection algorithm outlined in [25] was initially focused on for implementation. This part was tested in software using MATLAB. While ensuring accurate detection was a goal of the implementation, the major focus lay in the hardware architecture used for

implementation. Some important definitions related to the detection algorithm and the algorithm are described here.

5.3.1 Sample entropy

Sample entropy is a useful tool for investigating the dynamics of the heart rate and other time series. Sample entropy is the negative natural algorithm of an estimate of the conditional probability that subseries (epochs) of length *m* that match point-wise within a tolerance *r* also match at the next point. The algorithm builds up a run of points matching within the tolerance *r* until there is not a match and keeps track of template matches in counters A(*k*) and B(*k*) for all lengths *k* up to *m*, a value chosen to be 1 in this particular implementation. If a particular run ends up being of length 4, for example, then that means 1 is added to the count for the template matches of length 4. The algorithm starts by finding all points that match the first point within a tolerance *r*. If points after those with runs of length 1 match the second point, the runs are now of length 2; otherwise the run is ended. This procedure of finding runs is continued until the end of the data.

5.3.2 Artificial neural network (ANN)

An artificial neural network is a mathematical computational model that simulates the structure or functional aspects of biological neural networks. It consists of an interconnected group of artificial neurons and processes information using weighted-connections between the nodes. In most cases, it is an adaptive system that changes its structure (weights) based on information that flows during the learning phase. Neural

networks are one of the most common machine learning techniques and are used in non-linear statistical data modeling tools. They can be used to model relationships between inputs and outputs to find patterns in data. For more information on sample entropy and neural networks, refer to [26].

5.3.3 Description of Algorithm

The data from a study conducted by the Department of Epileptology, University of Bonn, Germany [27], was used for the verification of the algorithm. The data consists of 200 sample sets with 100 datasets of seizure patterns and 100 datasets of normal EEG patterns. Of these 200 datasets, 60 each from the seizure set and normal set are used for training the neural network and the remaining 40 sets each are used for the detection purpose. The datasets were collected using a 12-bit ADC with a sampling frequency of 173 Hz.

The sample entropy serves as the preprocessing element to reduce the sample size that serves as the input to the neural network. If a neural network of any size could be supported, the perfect neural network would have the number of input neurons equal to the size of a window of data. But this is not a feasible approach. Hence, the sample entropy helps in extracting a feature of a window of data that can be used as a classifying property in the neural network classification process.

Training of the neural network was achieved using Multiple Back-Propagation software [28]. The software helps to train a specified network by providing the training data and test data as the inputs along with the size of the neural network, i.e., the number

of neurons in the input, hidden and output layers. Training can be determined to be sufficient when the mean squared error is reduced to a minimum. At that point, the weights of the neurons / connections between all the neurons can be obtained. The software is also capable of generating equivalent C code.

A neural network with 4 input neurons, 3 hidden neurons and 1 output neuron was selected as this provided a small enough neural network for an optimized implementation in hardware with high detection rate. So 1 second of data from the dataset was divided into 4 windows having approximately 42 data points, which gives us an array of 4 sample entropy values, one for each window. As noted earlier, 120 datasets (60 seizure datasets, 60 normal datasets) are used for training. A value of 1 from the output neuron served as the seizure indicator, and 0 otherwise. The weights for the neural network are then obtained as the output of the training process. The training was determined to be sufficient when the mean squared error value reached close to 0.03. These weights gave superior results during detection. The entire algorithm can be depicted as in Figure 6.

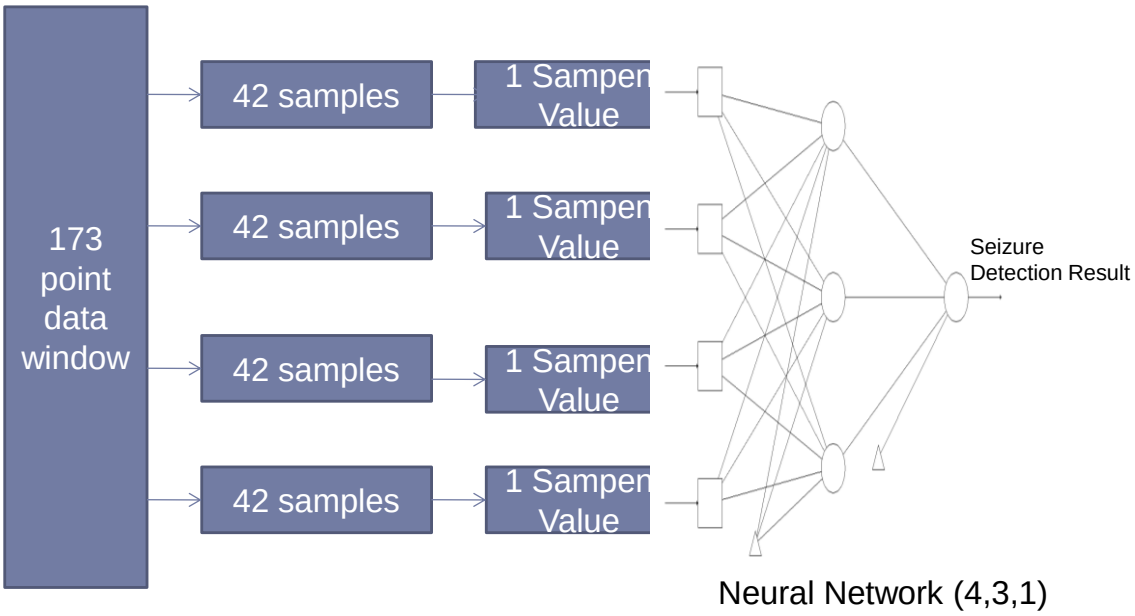

Figure 6: Seizure Detection using Sample Entropy and Neural Network

5.4 Variation of the Algorithm

It was noted during the implementation stage of the sample entropy based algorithm that the feature extraction process involved using a feature that was able to map the difference between normal and abnormal data to a larger range, which showed sufficient amount of difference in the majority of the cases so that the neural network could map that value to a binary output. While sample entropy provided a very high detection, it was also seen that the heart of the sample entropy module relied on one particular inequality relation which can be written as

$$| \text{sample}_1 - \text{sample}_2 | < k * \text{standard deviation (window of samples)}$$

where sample_1 and sample_2 are two samples within a window of samples and k is constant. In the case of seizure data, the variance of values within a particular window is large while the difference between samples within that window is small, i.e., more points

in that window will satisfy the above given inequality. In the case of normal EEG data, the variance of the entire window will be small while the difference between samples of the window will be large, which causes fewer points to satisfy that inequality. Thus, there exists a possible mapping of the set of points to either the seizure or normal category based on the value of the variance of a window of samples. This thesis was initially tested out in software and based on the results obtained; a hardware implementation was carried out. The accuracy of the detection scheme depends on the ability of the neural network to find weights which reduce the mean squared error. Both sample entropy and variance gave weights that had a approximate root mean square error of around 0.03. So the only difference lies in the hardware implementation of the two logic modules. Two implementations were carried out to determine the most optimal one. In one, the sample entropy was used as the feature for the input to the neural network, and in the other, the variance module's outputs were used as inputs to the neural network.

5.5 System Implementation and Operation

The overall operation of the system can be explained as follows. The seizure detector module (SeDM) is the algorithmic implementation of the seizure detection algorithm in the hardware.

The SeDM is internally composed of two structures – the sample entropy module, or the variance module, and the neural network module. The weights which were computed during the training process of the neural network are directly embedded within the neural network module. For verification purposes, the remaining 80 datasets from the database,

which were not used for training, are used for testing. The testing process is meant to mimic the real life situation where data is sampled and obtained from an ADC. This sampled data will then be fed to the SeDM. However, for testing purposes, we only focus on the processing part that follows the sampling part.

To mimic the reception data/collection of sampled input signals from an ADC, a ROM which is preloaded with data from the Bonn database is connected to the SeDM. It is also possible that all the data from the ADC is not intended for the SeDM. To this end, an encoding is used. The dataset requires a resolution of 12 bits to represent signed values from -2048 to 2047. An additional 4 bits are used to indicate that the data from the ROM is for the SeDM. These 4 bits are set high, i.e., a binary value of "1111" or 0xF. Four bits are used to provide support for future additional features like configuration of the SeDM to indicate the size of the input buffers required for supporting an increase in sampling rate, if need be. Also, the addition of 4 bits makes the data the same size as that of a short data type in a software implementation. An example depicting the encoded data is as follows:

$$\text{Sampled value} = 100_{10} = 0\text{x}064_{16} \Longrightarrow \text{stored as } 0\text{xF}064_{16} \text{ in ROM.}$$

A few screenshots from the Modelsim simulation of the module's operation are shown in Figures 7-10. The screenshots are from the implementation of the variance module in hardware. The signal labeled “outneuron” shows the output of the neural network, and the array of signals labeled “variance” shows the variance of a particular

window of data. One second of data consists of 173 data points which are divided into 4 windows of 42 points and the variance within each window is calculated. High values of variance represent seizure data, and the output of the neural network will be a 1. Low or near-zero values represent normal data, and the output of the neural network is a 0. The neural network output is a 30 bit fixed point value, with the lowest 16 bits representing the fractional part and the upper 14 representing the integer part. So if the first 4 digits are 0001, it indicates an integer value of 1. For example, Figure 7 shows variance values of 9, 28, 12 and 28 which are inputs to the neural network, while Figure 8 has zero as its 4 variance values. Figure 7 represents seizure data and Figure 8 represents normal data. So the output neuron (outneuron) in Figures 7 and 9 shows a value of 1, while in Figures 8 and 10 it shows a value of 0.

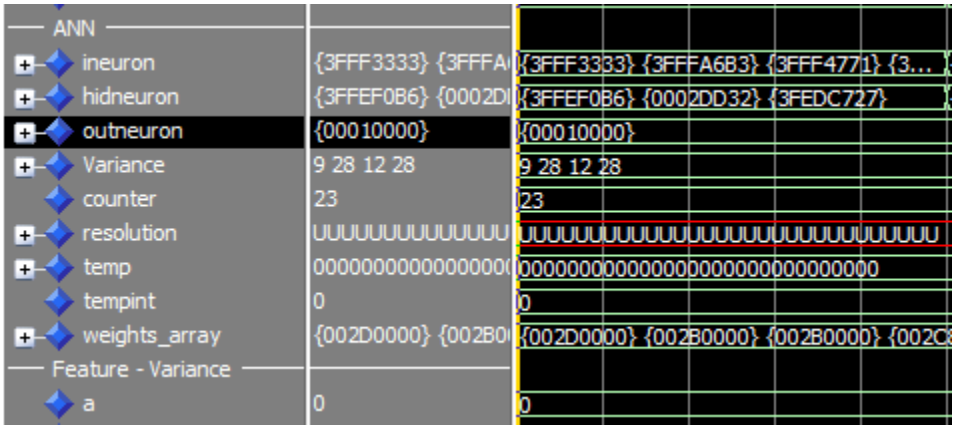

Figure 7: Variance and Detection - Seizure - Data Set 1

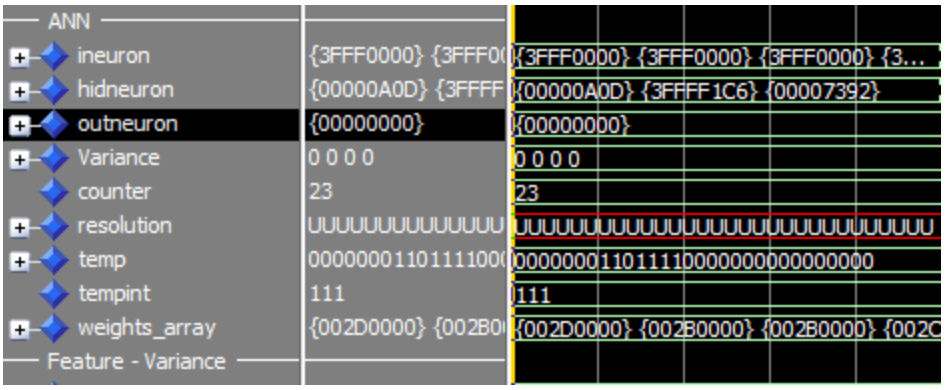

Figure 8: Variance and Detection - Normal - Data Set 2

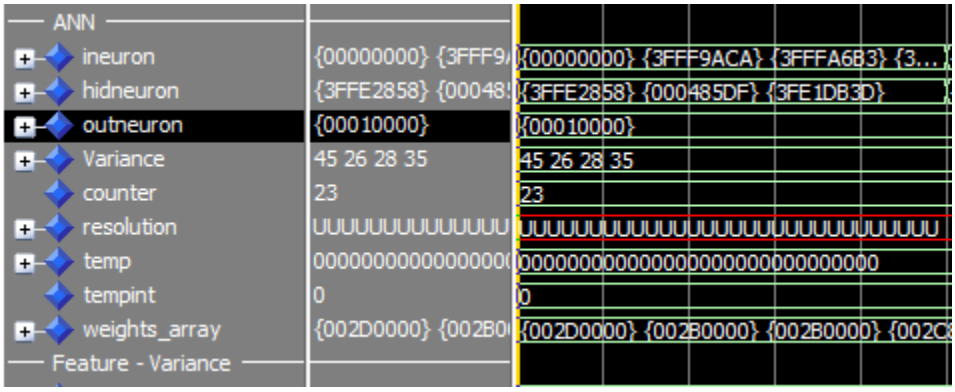

Figure 9: Variance and Detection - Seizure - Data Set 3

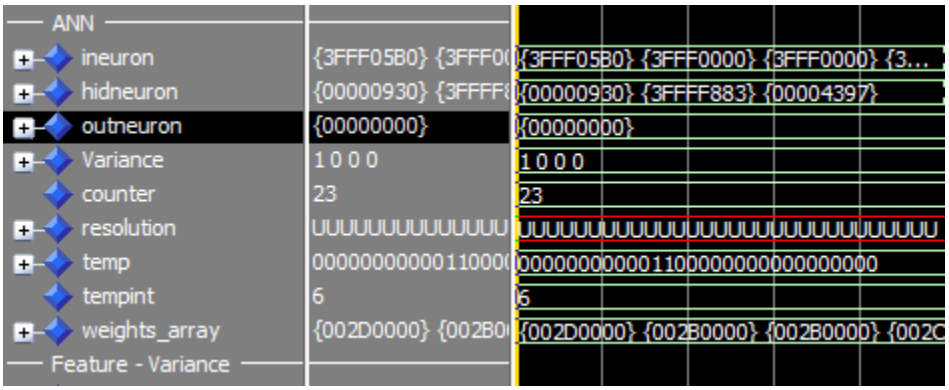

Figure 10: Variance and Detection - Normal - Data Set 4

5.6 Preliminary Synthesis Results

Although the final synthesis and hardware implementation was not carried out, early synthesis results showed the logic usage summarized in Tables 5 and 6 when synthesized for the Altera Stratix II platform (EP2S60F2673). The total number of ALUTs on this device is 48352.

Table 5: Logic Usage for Sample Entropy Module based SeDM

Module Name	ALUT Usage	Percentage (%)
Sample Entropy Module	36964	76.44
Neural Network Module	15432	31.91
SeDM (Total)	52396	108.35

Table 6: Logic Usage for Variance Module based SeDM

Module Name	ALUT Usage	Percentage
Variance Module	1975	4.084
Neural Network Module	15432	31.910
SeDM (Total)	17407	35.994

It can be seen that the sample entropy module requires significantly more resources than a simpler variance module, both of which offer the same level of detection accuracy.

The fact that the SeDM with sample entropy requires over 100 % of the resources available on the board should not be perceived as a limitation of the scheme because the FPGA fabric used for implementation was quite small. On the cutting-edge FPGA devices like Stratix IV platform from Altera and Virtex 5 from Xilinx, this logic amounts to approximately 30% of the entire FPGA chip's resources.

However, based on the usability of the algorithm for EEG seizure detection, it has to be concluded that the detection scheme based on variance of sample values offers an efficient hardware implementation with a good detection rate; therefore, the SeDM implementation with the variance module and neural network is chosen as the final detection scheme.

5.7 Discussion

It can be seen that the sample entropy module requires a significant amount of logic when implemented in hardware. This is because of its inherently sequential nature of computation that requires a significant amount of sample history to be stored in buffers; i.e., a certain number of previous samples are required for computing the sample entropy and this occupies logic. Also, the sample entropy value requires the presence of a nested loop which exhibits loop counter dependence and, hence, destroys any parallelism that can be extracted from it. Several optimizations were carried out to reduce the resource usage to the numbers above which included loop unrolling and elimination of all floating point computation. This made it easier for the Quartus-II software to carry out optimizations when synthesizing. On the other hand, the variance module required a

significantly smaller logic count for onboard implementation because of the absence of the loops and presence of rather simple operations.

The neural network module requires the usage of number representation with as much precision as possible. While a software implementation provides the advantage of floating point usage, it is not possible in hardware. A 30 bit fixed point implementation was used. The linear activation function for the neurons in the hidden layer helped avoid the use of an exponential sigmoid function, although it could not be avoided in the output layer.

CHAPTER 6

RELIABLE BIOMEDICAL PROCESSING ENGINE

The FPGA-based hardware implementation of a seizure detection algorithm provides the foundation for a biomedical processing engine that is capable of using these embedded modules for analyzing and making conclusions using real-time data. Though hardware implementation of the sample entropy computation algorithm proved to be cumbersome for EEG seizure detection, sample entropy has shown to be useful for detecting irregularities in several other time series such as heart rate variability. Hence, it will form an important part of a module that is capable of multi-feature extraction based monitoring. In addition to these techniques, there are several well known and traditional signal processing techniques which can be applied in such biomedical systems, including fast fourier transform (FFT), auto-correlation, cross-correlation, statistical testing, power spectral density, machine learning based classification etc. While these techniques are not completely new, having a hardware unit that is capable of supporting such computation is a novel idea, and implementing these algorithms exclusively in hardware will provide a significant performance boost to biomedical monitoring units which rely primarily on such techniques to differentiate between data from a healthy and from a sick individual. Described here is a framework for a *Reliable Biomedical Processing Engine (RBPE)*, a processing unit that is capable of supporting the above-mentioned computation directly in hardware to cater to the needs of a specific application domain.

6.1 Working of the Reliable Biomedical Processing Engine

Figure 11 shows a block diagram of the engine. Its main component is the interface with the source of data or the ADC. A typical ADC usually has several channels which are multiplexed to one sampling unit. Assuming there is more one than one channel to be sampled and there is a different kind of processing to be performed on each channel, the *Encoding Unit (EU)* helps achieve this goal. It is capable of communicating with the higher level software to receive the following pieces of information:

1. Settings for each of the modules – like a buffer size for a module, a sampling rate, a constant needed for one module's algorithm, etc.
2. An annotation which indicates the module for which the sampled data is intended; e.g., if channel 2 has been selected to be processed with variance module, we want the variance module to only process data from channel 2.

This can be achieved by tagging the data received with a certain number of bits which will indicate which module will process the data. For example, assuming 4 bits are used to tag the data, a value of 0xA could indicate that the data is for the sample entropy module. The same tag can be also have a particular pattern to indicate that multiple units need to process data from a particular stream. This feature can be used to implement reliability of computation by means of redundancy in the feature extraction process. The same data stream can be operated on by two or more feature modules and their outputs can be compared to facilitate the decision-making process.

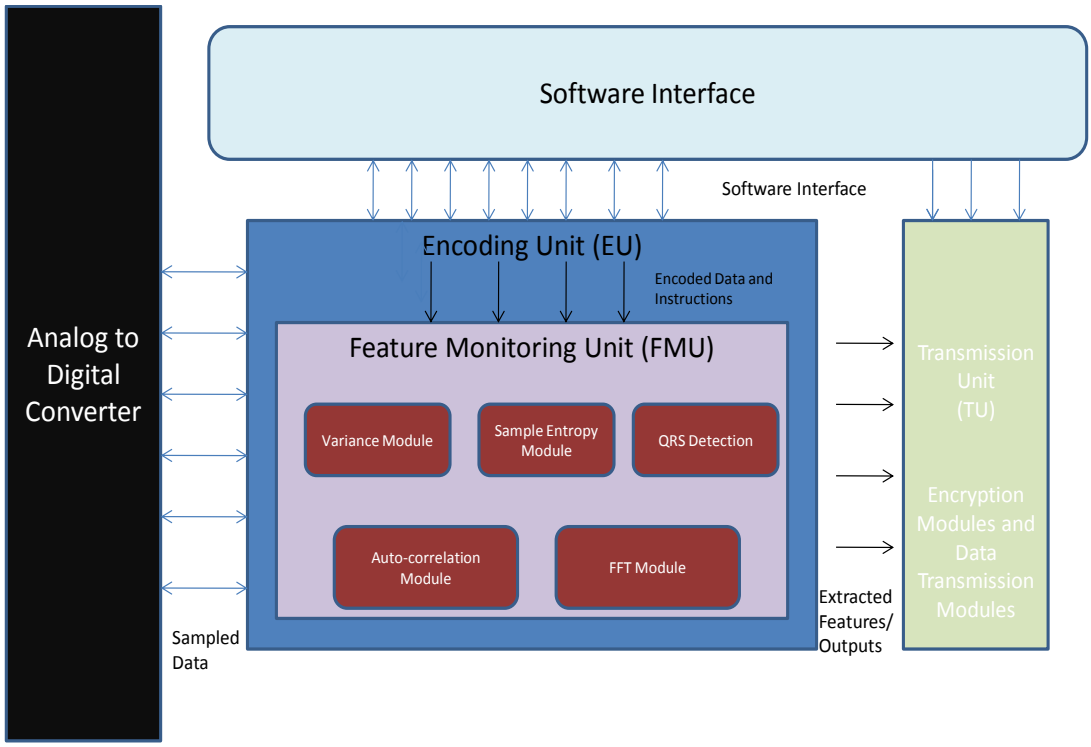

Figure 11: Architecture of a Reliable Biomedical Processing Engine

The heart of the engine is the *Feature Monitoring Unit (FMU)* which monitors a particular feature for a particular data. To enable multiple data streams use the same feature, it is possible to use extra tag bits with the data, i.e., additional bits to encode the data stream information. The multiple modules shown here can also be connected via a common communication bus like the AMBA APB bus. This provide a "plug and play" interface, which helps add or remove feature modules as and when needed. This provides an advantage over including these modules as part of a pipelined execution unit. The output of the processing unit is either a prediction or the value of a processing. It is also important that the same tag bits be appended to the data that the module is processing so that the output can be classified too. The Collection Unit is used to write / store the data.

The final stage of the module is the *Transmission Unit (TU)* which provides the additional features like encryption and transmission of data and other security features. Advanced Encryption cores like AES, RSA, etc., can be directly embedded here.

In simpler terms, the operation of the Reliable Biomedical Monitoring Engine can be thought of as analogous to a pipelined CPU. The Instruction Fetch, Decode, Execution, Memory and Write Back stages are now replaced with Sample, Decode and Execute and Output stages. However, the difference between the two lies in the application space. The former is used for general purpose computing while the latter is more or less tailored for signal processing in a biomedical application domain.

6.2 Implementation of the Reliable Biomedical Processing Engine

In this section, a simulation-based implementation the RBPE is discussed. The main goal of the architecture is to provide flexibility and ease of development of the end design with a focus on biomedical monitoring. This helps the user to focus on development of algorithms for detection rather than the platform itself. The support for specialized operations in hardware reduces the software latency involved in computation. Additionally, the user should be able to add support for more operations in hardware if required. For example, in the implementation of the SeDM described earlier, another algorithm that uses a variance and neural network computation cannot use the modules in the SeDM because of the coupling between the two modules. Thus, the SeDM implementation is tailored towards a single-cycle operation execution in an ASIC while the RBPE is suitable for initial prototype development in FPGA logic or as a final SoC.

Keeping these design requirements in mind, the RBPE is implemented as a soft IP core within an SoC design featuring a general purpose processor to support the algorithm in software and essentially control the communication between the various hardware modules.

The engine has been implemented by integrating it with the open-source Leon3 soft-core processor, provided by Gaisler Research [29]. The Leon3 is a 32-bit synthesizable processor core based on the SPARC V8 architecture. The core is highly configurable and particularly suitable for SoC designs. It is a new implementation of the architecture with a 7-stage pipeline and multiprocessor support. It is distributed as part of the GRLIB IP library [30], a library of several configurable IP cores. The library includes cores for the Leon3 processor, memory controllers for PROM, SRAM and DDR/DDR2 memory and several other peripherals like SPI, CAN and UART.

A very important component supplied as part of this library is the AMBA AHB/APB controller cores. AMBA was introduced by ARM [31] and is one of the most widely used on-chip buses for SoC designs. The main components of the specification are the AHB and APB buses. The AHB is a single-edge clock bus protocol designed to support several bus masters with the ability to do carry-out burst and pipelined transfers in addition to a single-cycle bus master handover. The APB bus is designed for low bandwidth control accesses such as register interfaces on system peripherals. This bus has an address and data phase similar to the AHB, but a much reduced low complexity signal list and lack of features such as burst transfers.

Gaisler Research provides templates for SoC designs for development on kits supplied by various FPGA board manufacturers. While the templates vary slightly in instantiation of the cores, the overall development process and IP cores used in the template are the same. The Leon3 processor is connected as a master to the AHB bus and is generally the only master, in a single processor based design. In a multiprocessor design, all instantiated Leon3 processors will be masters on the bus. For debugging purposes, a debugging unit can also be connected as a master. The AHB controller serves as the arbiter, bus multiplexer and slave decoder according to the AMBA standard. The generation of the slave-select signal is done using a "plug and play" method supported by all IP cores in the library supporting the AHB interface. The essence of the plug and play operation is a configuration record identifying a certain core specification, which is driven on the bus and decoded by the AHB controller. The memory controller is also realized as a slave on the bus. It combines the controller for the PROM, SRAM and DRAM memory. Other high speed devices and peripherals such as USB, Ethernet and CAN modules can also be instantiated as AHB slaves. The APB bus is connected to the AHB bus through the AHB/APB bridge which serves as a slave on the AHB bus and is the master on the APB bus. The APB bus is used to interface devices like timers, interrupt-controllers and I/O port interfaces.

For this implementation, the RBPE components are instantiated as peripherals connected to the system on the APB bus. The two modules comprising the SeDM are implemented as part of the RBPE. Figure 12 shows the integration of the RBPE with the Leon3 processor.

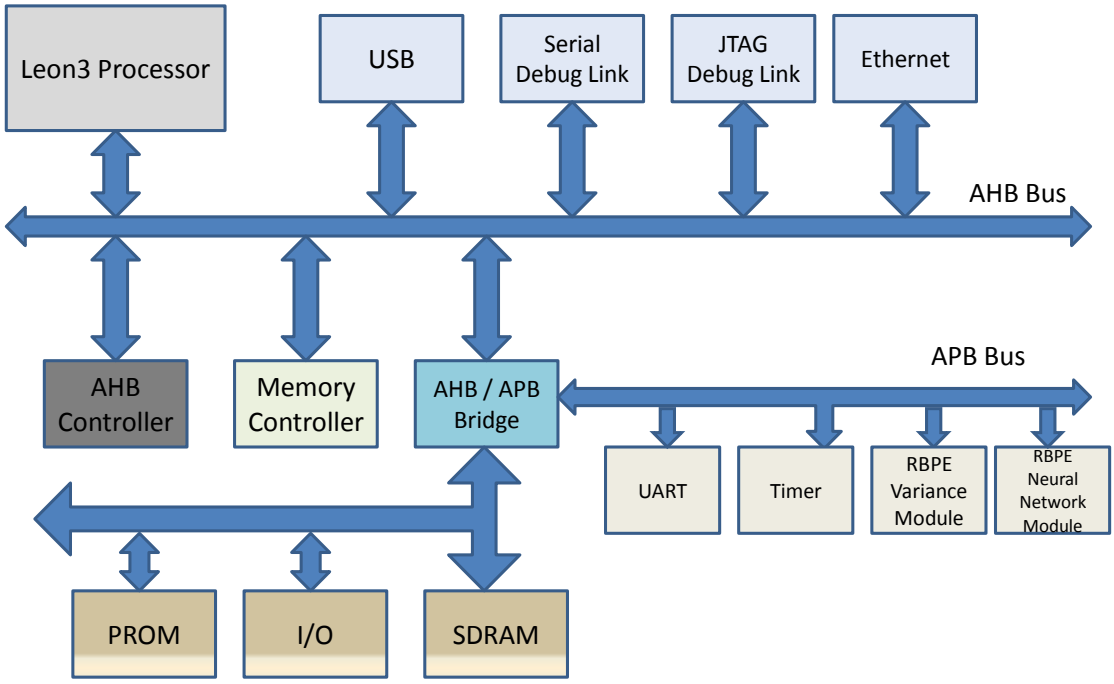

Figure 12: RBPE Modules Integrated with Leon3

The application consists of the algorithm written in C that is run on the Leon3 processor. Since the intent of the engine is to detect abnormalities in analog signals, a detection algorithm will read the digital data sampled by the on-chip ADC and send them to appropriate modules in the RBPE. The result of the computation in each module can be read out. The application retains control of the various operations as it is responsible for chaining the outputs of a computation with the input of another module. Figure 13 shows the operation of the entire system and how the various modules can be used to implement a desired detection scheme. However, in the simulation, only the part following the sampling stage is implemented.

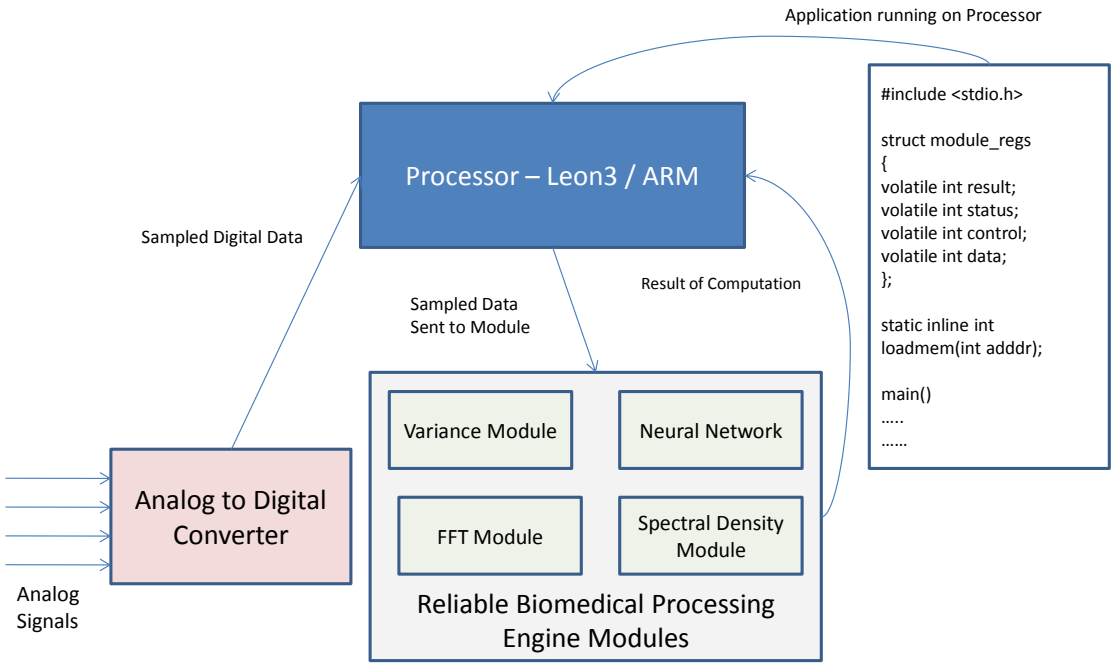

Figure 13: Workings of the RBPE

The implementation ties in with the generic architecture of the RBPE described in the beginning of the chapter in the following ways. It was proposed to have an encoding unit to annotate the data to indicate the module for which it was intended and to configure the modules itself. In this implementation, it is achieved using a combination of both hardware and software. When instantiating the modules on the APB bus, each module has a unique address space allocated to it and any location within that space can easily be recognized by that module. Therefore, additional annotation is unnecessary. Configuration of the modules is carried out by the application at the software level. Although the application is required to know some details about the underlying hardware, this is necessary to allow flexible configuration of the modules. The feature monitoring unit described consists of the various modules instantiated on the APB bus. The

transmission unit consists of the various other peripherals included on the APB bus such the UART.

The values for the various parameters in the registers described in the variance and neural network modules are to ensure optimal results using data in the Bonn Database.

6.3 Variance Module

In this section, the architecture, functionality and operation of the RBPE variance module are explained. The variance module is used to compute the variance of a sequence of samples. The functionality, while the same as the module in the SeDM, differs in the implementation. The SeDM was designed to compute the variance with no ability for configuration of features. However, this implementation allows application-level configuration of the important features of the module along with ability to communicate the status of the computation to the application. This is achieved using registers mapped to the address space.

Five registers are included within the module to support configuration and status information transfer between the main processor and module. All registers are 4 bytes or a word long. Bits in each of the registers are numbered in little-endian format with the least significant bit being bit 0 and the most significant bit being bit 31. The 5 registers are - variance result register, status register, control register, data register and scaler register. The functions of the registers are as follows:

1. Variance result register

The register is mapped to address 0x00 within the address space of the module. It is a read-only register and holds the result of the last variance computation. It is initialized to zero on a reset. Figure 14 shows the structure of the register.

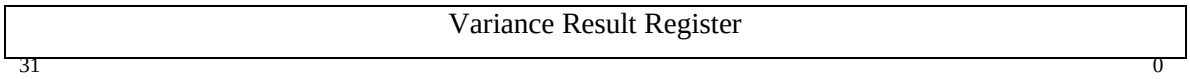

Figure 14: Variance Register

2. Status register

The register is mapped to address 0x04 within the address space. It is a read-only register and holds information about the status of the computation within the module, which can be polled by the application. For now, only a single bit is used which indicates if the variance computation is complete. It is set to 1 when the variance computation is in progress and is zero otherwise. Bit 1 serves this purpose and is labeled “vcomp.” The other bits are reserved for future use. Figure 15 shows the structure of the register.

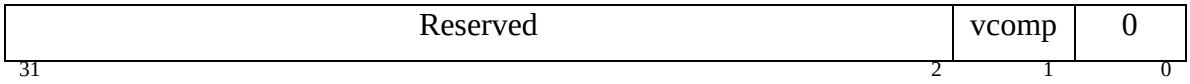

Figure 15: Status Register

3. Control register

The register is mapped to 0x08 within the address space. It is a write-only register. Bit slice [7:0], labeled as “samplecount” is used to indicate 1 less than the number of

samples for which the variance has to be computed. This is variable and can take a value from 0 to 254, making the range of the sample size 1 to 255. Since variance computation involves the use of the square of the number of samples, 16 bits are used to load the value into this register. Bit slice [24:8] is used for this purpose and is labeled as "samplecountsq." It can take a value from 1 to 65525. To ensure correct variance computation, the application must load samplecountsq with the square of the value loaded into samplecount. Figure 16 shows the structure of the control register.

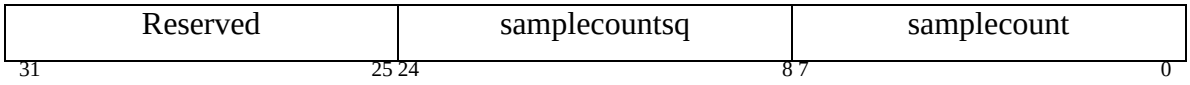

Figure 16: Control Register

4. Data register

This register is mapped to 0x0B of the address space. It is a write-only register and retains the value of the last sample that was loaded for variance computation. It is indicated as "lsample." While not particularly useful for functionality purposes, it serves as a useful aid for testing. To test if the module is functioning properly, the value loaded can be read back to check for equality. The structure of the register is shown in Figure 17.

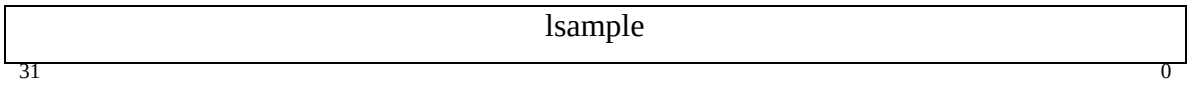

Figure 17: Data Register

5. Scaler register

Figure 18 shows the structure of the scaler register and it is mapped to 0x10 of the address space. It is a write-only register. It is represented as “scaler” and is used to hold a value by which the variance result is scaled down. If scaling is not desired, a value of 1 must be loaded. Bit slice [15:0] represents the value of scaler. Bit slice [31:16] is reserved.

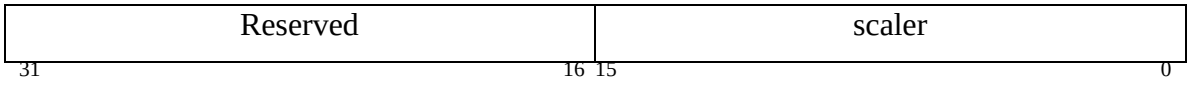

Figure 18: Scaler Register

6.3.1 Operation of the variance module

The module is instantiated as slave 7 on the APB bus with a base address of 0x80000700. The application is responsible for configuring the module, loading the parameters necessary for operation and reading out the result. Each of the above is examined in detail. Snippets of code with appropriate comments to perform each of the operations described here are included in Appendix A.

6.3.2 Configuration of the module

The configuration of the module is carried by loading the relevant parameters of the control register. For this purpose, the following parameters are loaded:

samplecount : 41

samplecountsq : 1764

scaler : 10000

The samplecount parameter takes a value of 41 because the variance has to be computed for a set of 42 samples. Thus, the samplecountsq parameter takes a value of 1764. The scaler value is used only because the SeDM implementation in Section 6.5 was also scaled down by a value of 10000. If the value of variance is not required to be scaled, a value of 1 should be loaded.

6.3.3 Loading data into the module

To load data into the module, the data register, which is present at an offset of 0x0B, is written with the value. The data is written sequentially until the limit set by the value of samplecount is reached.

6.3.4 Computation of result

Once 42 samples have been loaded, the application polls the full bit of the status register to see when the variance computation is complete. The present implementation utilizes 4 clock cycles to compute the variance of the entire loaded sample set.

6.3.5 Reading the result

The result register contains the result of the variance computation. It can be obtained by reading the result register, which is mapped to address offset 0x00.

6.4 Neural Network Module

The neural network module realizes the neural network described as part of the SeDM implementation. However, like the variance module in the RBPE, this implementation differs from the one in the SeDM as it supports configuration of the module using the application. The neural network is capable of supporting the following numbers of nodes in the various layers:

Number of layers	:	3
Number of hidden layers	:	1
Number of nodes in the input layer	:	1 - 4
Number of nodes in the hidden layer	:	1 - 4
Number of nodes in the output layer	:	1

For the purposes of the seizure detection algorithm in the SeDM, it was seen that a neural network with the above parameters is sufficient. There are 5 types of registers in the module. They are weight registers, control register, input registers, result register and status register. The addresses for the registers are relative to the base address of the module. It was instantiated as slave 6 on the APB bus, giving it a base address of 0x80000600 with 256 bytes within the memory space. As some constraints are dependent on the number of nodes in the input and hidden layers, for the rest of the description it is assumed there are *i* input nodes and *h* nodes in the hidden layer.

1. Weight registers

These registers are used to hold the weights of the connections between the various nodes in the network. The weights, generated by multiple back-propagation software, are used here. Since they are in a specific format, the weight array is also assumed in the same format to ensure compatibility. The weights have to be loaded in the following format to ensure correct operation of the module.

First *i* locations : Weights for the *i* nodes of the network

Next (4-*i*) locations : 1 if $i < 4$

Next ($h + h * i$) locations : Weights for the *h* nodes of the network

Next (h +1) locations : Weights for the output node

For the hidden nodes, there are (i + 1) weights for each of the *h* nodes, and they are intended to be used successively by the hidden nodes. The weight registers for the input nodes which are absent, are to be loaded with a value of 1, and the weight registers for the hidden nodes which are absent are to be loaded with a value of 0. These registers are mapped to the address range 0x00 - 0x7F. To reduce the hardware overhead, the weights have to be in 30 bit fixed-point format with a 14-bit integer part and a 16-bit fraction part. Thus, the conversion to fixed-point within the module is avoided. They are implemented as write-only registers. The weight is referred to as "Weight" and is shown in Figure 19. The total number of weight registers required is given by $(i + i*h + 2*h + 1)$. It is 29 in this implementation where $i = 4$, $h = 4$. If a neural network with larger number of input and hidden layer neurons is desired, additional weight registers will have to generated.

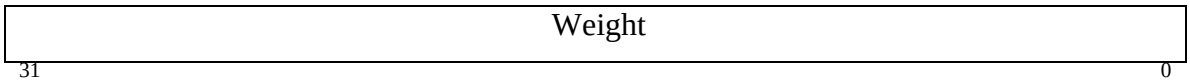

Figure 19: Weight Register

2. Control register

This register is mapped to address 0x80. It is used to load the value of 1 less than the number of input neurons and hidden neurons, referred to as “icount” and “hcount” respectively. Bit slice [1:0] represents icount and bit slice [3:2] represents hcount. It is a write-only register and its structure is shown in Figure 20. If a neural network with more than 4 input neurons and 4 hidden layer neurons are required, additional bits will have to be allocated for icount and hcount.

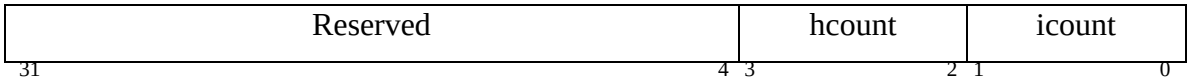

Figure 20: Control Register

3. Input register

This register is mapped to 0x84. It is used to load the value of input neurons. There are 4 input registers as the input layer has 4 neurons. These values have to be loaded sequentially to ensure correct operation; i.e., the first value loaded into this address is taken to be the value of the first input neuron. As the result of the variance computation is an integer, the value loaded into this register is also assumed to be an integer. The

conversion to fixed-point is carried out in hardware. Figure 21 shows the structure of the register and the value is represented as “input.” It is a write-only register.

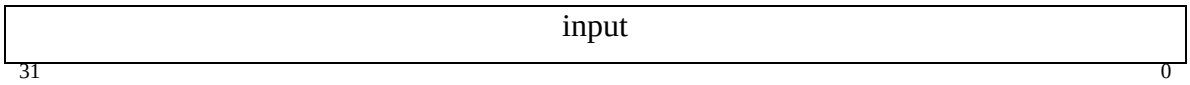

Figure 21: Input Register

4. Result register

This register is mapped to 0x88. It is used to hold the result of the neural network computation or the value computed by the output node of the network. It is a read-only register and is represented as “Result.” It should be remembered that the result is a fixed-point number. The required bits can be extracted by means of a shift operation to check for a specific condition. Generally, only the upper 16 bits will be required and can be obtained by a simple typecast from within the application. However, it should be remembered that the two most significant bits from the 16 bits will be zeros on account of the 14-bit integral part of the fixed-point format. Figure 22 shows the result register.

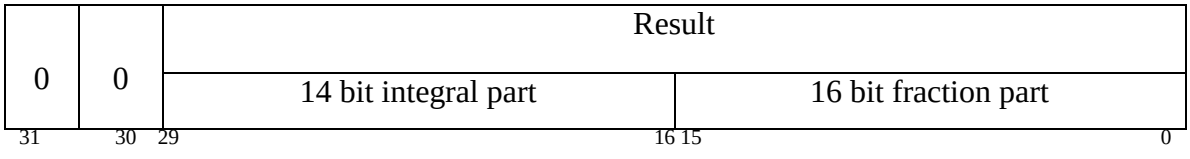

Figure 22: Result Register

5. Status register

This register is mapped to 0x8C and represents the status of the module. The least significant bit, represented as “rcomp,” represents whether the module has finished computing the output. This bit is raised high when the computation is in progress and lowered when complete. It can be polled to see if the computation is complete and then the result can be read. The other bits are reserved for future revisions. Figure 23 shows the structure of the status register.

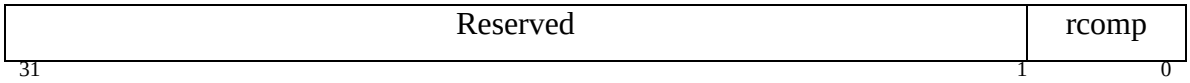

Figure 23: Status Register

6.4.1 Operation of the neural network module

As mentioned earlier, the module is instantiated as slave 6 on the APB bus with a base address of 0x80000600. The application is responsible for configuring the network, loading the parameters necessary for operation and reading out the result. Each of the above is examined in detail. Snippets of code with appropriate comments to perform each of the operations described here are included in Appendix B.

6.4.2 Configuration of the module

The configuration aspect of the module involves loading the weights into the weight registers. The rules specified in the explanation of the weight registers must be followed. The weights are in fixed-point format. This is followed by loading the control register

with the specified numbers for input and hidden layer neurons. Since the SeDM scheme utilized a network with 4 input neurons and 3 hidden layer neurons, the following values are obtained for icount and hcount:

icount : 3

hcount : 2

6.4.3 Loading the input neuron values

To start the computation, the input values for all the input neurons have to be loaded sequentially to ensure correspondence with the weights in the weight registers. Once the number of inputs written to the input register equals the value represented by icount, the computation is started.

6.4.4 Computation of result

Once the input neurons are loaded, computation is started and the rcomp bit in the status register is raised high. The module will take $(i + h + 2)$ clock cycles to complete. The rcomp is lowered once the computation is complete. The application can poll this bit to check for the status.

6.4.5 Reading the result

The result can be read out by loading from the result register. The output is in fixed-point format, and since for seizure detection the output is a 1 or 0, with 1 indicating the presence of a seizure and 0 otherwise, only the least significant bit in the 14-bit integer

part is needed. A simple bitwise AND operation followed by a logical right shift can be used to obtain that bit.

6.5 Combined operation of the Variance and Neural Network Modules

The functioning of the neural network module in conjunction with the variance module is explained here.

The variance module's control register is loaded with the parameters as mentioned in Section 7.2.2. This is followed by initializing the neural network's control register and loading the weights into the weight registers, and this is followed by writing to the data registers of the variance module with the sample values. Once the number of samples equal to the value of samplecount has been loaded, the variance is computed and the variance can be read from the result register of the variance module. This can be immediately written to the input register of the neural network, or it can be stored in memory before loading all the input neurons sequentially. Once all the variance values have been obtained and written to the input register of the neural network, the neural network begins computation. The rcomp bit is polled from within the application. Since the neural network has 4 input neurons, 3 hidden neurons and 1 output neuron, it takes 9 clock cycles to complete, after which the output can be read from the result register. Bit 16 of the result is checked for a high or a low value. A high is identified as a seizure and a low is identified as normal EEG data.

6.6 Simulation Environment

The application is compiled using the SPARC BCC cross-compiler provided by Gaisler Research. The RTL level simulation is carried out using Modelsim. For simulation, the application is compiled and the output of the compilation process is converted into a PROM boot loader along with the application executable. The executable file is loaded into RAM. The data is loaded into the module using the application. Figure 24 shows a screenshot of the simulation log obtained in Modelsim.

Figure 24: Screenshot of Simulation Log

6.7 Evaluation of the Seizure Detection Algorithm

As mentioned in the SeDM implementation, the datasets used were obtained from the Bonn database. It is necessary to evaluate the performance of the algorithm. The values of standard statistical measures are calculated to analyze the quality of the algorithm. The measures used are briefly explained here.

Specificity - Specificity measures the ratio of negatives which are correctly identified. Again, this includes true negatives and false negatives. In this case, true negative represents the number of normal EEG patterns identified as non-seizure patterns and false negative represents the number of seizure EEG patterns identified as healthy. A high specificity shows that there are low type-I errors; i.e., most healthy EEG patterns are identified as healthy.

Sensitivity - Sensitivity measures the ratio of actual positive cases that are identified by a classification test. This includes the number of true positives and false positives. In this case, the true positive represents the normal EEG patterns identified as normal and the seizure patterns identified as abnormal. A high sensitivity shows a low type-II error rate; i.e., most unhealthy seizure patterns are identified correctly.

F-measure - F-measure can be used as a single measure of performance. The F-measure is the harmonic mean of sensitivity and specificity. It is a very common measure used in statistical testing and information retrieval. In information retrieval, specificity is also

called precision and sensitivity is also called recall. F-measure is sometimes referred to as F-score.

Accuracy - Accuracy is a measure closely related to the F-measure. It is defined as the ratio of the correctly identified cases to the total number of cases. It shows the number of correct outputs and it includes the true positives and true negatives.

The experiment carried out can be explained as follows. From the 100 datasets of normal and seizure data, 60 sets each are used for training the neural network. Thus, if these sets are provided as inputs to the two modules, the detection rate will be close to 100%. Therefore, the remaining 40 sets are used for testing purposes. The same computation is performed on MATLAB with floating-point precision and the expected results are known. The verification and evaluation of the algorithm are carried out in the application that runs on the Leon3 by comparing the computed output with the expected output known beforehand. Code to perform the same is included in Appendix C. Figure 25 shows a screenshot of the results computed in Modelsim.

```
# ahbctrl:       memory at 0x00000000, size 1 Mbyte, cacheable, prefetch
# ahbctrl: slv1: Gaisler Research        AHB/APB Bridge
# ahbctrl:       memory at 0x80000000, size 1 Mbyte
# ahbctrl: slv2: Gaisler Research        Leon3 Debug Support Unit
# ahbctrl:       memory at 0x90000000, size 256 Mbyte
# ahbctrl: slv3: Gaisler Research        Single-port DDR266 controller
# ahbctrl:       memory at 0x40000000, size 1024 Mbyte, cacheable, prefetch
# ahbctrl:       I/O port at 0xfff00100, size 256 byte
# ahbctrl: slv4: Gaisler Research        Test report module
# ahbctrl:       memory at 0x20000000, size 1 Mbyte
# apbctrl: APB Bridge at 0x80000000 rev 1
# apbctrl: slv1: Gaisler Research        Generic UART
# apbctrl:       I/O ports at 0x80000100, size 256 byte
# apbctrl: slv2: Gaisler Research        Multi-processor Interrupt Ctrl.
# apbctrl:       I/O ports at 0x80000200, size 256 byte
# apbctrl: slv3: Gaisler Research        Modular Timer Unit
# apbctrl:       I/O ports at 0x80000300, size 256 byte
# apbctrl: slv4: Gaisler Research        AHB Debug UART
# apbctrl:       I/O ports at 0x80000400, size 256 byte
# apbctrl: slv6: DEPEND Group UIUC       Depend Neural Network Module
# apbctrl:       I/O ports at 0x80000600, size 256 byte
# apbctrl: slv7: DEPEND Group UIUC       Depend Variance Module
# apbctrl:       I/O ports at 0x80000700, size 256 byte
# testmod4: Test report module
# ahbrom0: 32-bit AHB ROM Module,  92 words, 7 address bits
# gptimer3: GR Timer Unit rev 0, 8-bit scaler, 2 32-bit timers, irq 8
# irqmp: Multi-processor Interrupt Controller rev 3, #cpu 1, eirq 0
# apbuart1: Generic UART rev 1, fifo 8, irq 2
# ddrsp3: 64-bit DDR266 controller rev 0, 256 Mbyte, 90 MHz DDR clock
# ahbjtag AHB Debug JTAG rev 0
# ahbuart4: AHB Debug UART rev 0
# dsu3_2: LEON3 Debug support unit + AHB Trace Buffer, 2 kbytes
# leon3_0: LEON3 SPARC V8 processor rev 0
# leon3_0: icache 2*8 kbyte, dcache 2*4 kbyte
# clkgen_virtex2: virtex-2 sdram/pci clock generator, version 1
# clkgen_virtex2: Frequency 100000 KHz, DCM divisor 13/20
#
# ****** Running DEPEND Biomedical Processing Engine Application ********
# Depend Variance Module
# Depend Neural Network Module
# False Positives :2
# False Negatives :13
# True Positives :907
# True Negatives :965
# Test passed, halting with IU error mode
#
```

Figure 25: Screenshot of Simulation Log During Testing Phase

CHAPTER 7

RESULTS

Based on the experiment explained in Chapter 6, the results obtained are presented in Table 7. The values of the various measures can be calculated as shown in Table 8.

Table 7: Results of Algorithmic Testing Phase

Quantity	Value
True positives (TP)	907
True negatives (TN)	965
False Positives (FP)	2
False Negatives (FN)	13
Total	1887

Table 8: Statistical Measures

Measure	Expression	Value (%)
Specificity (Sp)	= TN / (TN + FP)	99.8
Sensitivity (Se)	= TP / (TP + FN)	98.6
F-measure	= 2 * Sp * Se / (Sp + Se)	99.2
Accuracy	= (TP + TN) / (TP + TN + FP + FN)	99.2

7.1 Explanation of Results - Algorithm

It can be seen that the quality of the seizure detection algorithm is quite high. It achieves almost 100 % specificity, which means that no healthy people are identified as having a seizure. This is quite important as it avoids triggering a false alarm and possible panic. The sensitivity is also quite high, which means that a very high number of seizures are correctly identified and a very small number of seizure patterns are missed by the detection scheme and identified as normal. A 99 % value for F-measure and accuracy confirms the suitability of the algorithm as a seizure detection scheme. Also, the algorithm uses a simple statistical measure like variance as the preprocessing feature, and this simplified the hardware implementation.

7.2 Explanation of Results - Hardware Implementation

Two implementations of the above seizure detection algorithm have been carried out. The hardware utilization for the SeDM is moderate and is suitable for an ASIC implementation as it will not meet the timing requirements when coupled with other modules as it has to be in the RBPE. The implementation of the algorithm as part of the RBPE provides the flexibility to integrate other modules and has low overhead. The computations are implemented as multi-cycle operations, which will reduce the hardware logic utilization. The variance module required only 4 clock cycles and the neural network only 9 to compute the result. Since the entire platform was only simulated, hardware utilization values are not available.

CHAPTER 8

CONCLUSIONS

This thesis has presented two possible hardware design styles that can be used for biomedical monitoring purposes with the key distinction between the two being the ability to chain complex operations with built-in hardware support in the FPGA scheme. In addition to the platform development, this thesis has put forward a novel seizure detection algorithm, which has a very high detection accuracy of 99.2 %.

There are a few key directions, at various levels of implementation, that can be focused on. The lowest level would be the physical layer or the layer that deals with the interconnection of sensors and processing elements so that a communication medium exists between the operating elements or nodes. Concepts used in ad-hoc networks can be implemented here to ensure there is a routing communication protocol among the different nodes. Ensuring connectivity with the sensors at all times is a very important issue. This involves the ability of the processing unit to differentiate between data obtained when the sensors are connected and data obtained in other cases. Since the application deals with sampled digital data, sampling channels with no inputs will also give a digital output. A possible solution to this problem would be to have a physical contact between the channel and the sensor whose connectivity can be polled by the application. The connection can be pulled to a logical high in one case and to a logical low in the other. Additionally, optical technology described in [32] could be used to overcome this issue.

Another direction at the hardware level would involve development using the RBPE. It provides a good framework in this regard and its design and functionality can be improved. To improve the functionality, other processing modules can be added to complement the existing variance and neural network modules. Hardware modules for other signal processing operations like power spectral density and correlation between signals could be developed.

At the software level, the focus can be directed more towards the algorithm that is used for monitoring and processing the data. Algorithms used for monitoring several different kinds of abnormalities can be investigated. It is necessary to have a number of good detection schemes to be able to justify the use of such a custom-made device. For example, the core unit in a seizure detection algorithm can be used side by side with a stroke or arrhythmia detection scheme to be able to monitor both EEG and ECG at the same time. In addition to these physiological signals, additional patient-related information can be collected such as blood pressure and motion information.

To summarize, the hardware approaches discussed in this thesis provide an excellent platform for multi-dimensional signal monitoring and feature extraction for use in biomedical applications to facilitate reliable and efficient decision making.

REFERENCES

[1] G.Z. Yang, *Body Sensor Networks*. London, England: Springer Books, 2006.

[2] R.K. Ganti, T.F. Abdelzaher, P. Jayachandran and J.A. Stankovic, "SATIRE: A software architecture for smart AtTIRE," in *Proceedings of 4th International Conference on Mobile Systems, Applications, and Services, Mobisys '06*, 2006, pp. 110-123.

[3] L.M. Hively, N.E. Clapp, V.A. Protopopescu, J. Joseph, C.E. Merican, and T. Lucht, *Epileptic Seizure Forewarning by Nonlinear Techniques*. Oak Ridge, TN: Oak Ridge National Labs, 2000.

[4] A. Shoeb, S. Schachter, T. Pang, and J. Guttag, "Implementation of closed-loop, EEG-triggered Vagus nerve stimulation using patient-specific seizure onset detection from scalp EEG," *Epilepsia*, vol. 48, S6, pp. 304, December 2007.

[5] A. Shoeb, S. Schachter, B. Bourgeois, S.T. Treves, and J. Guttag, "Impact of patient-specificity on seizure onset detection performance," in *29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society,* August 2007, pp. 4110-4114.

[6] N. Verma, A. Shoeb, J. Guttag, and A. Chandrakasan, "A micro-power EEG acquisition SoC with integrated seizure detection processor for continuous patient monitoring," *2009 Symposium on VLSI Circuits,* June 2009, pp. 62-63.

[7] B. Liu, Y. Zhang, Z. Liu, and C. Yin, "An embedded EEG analyzing system based on uC/OS-II," in *Proceedings of the 29th Annual International Conference of the IEEE EMBS,* August 2007, pp. 2648-2471.

[8] Y. Mendelson. (2009, June). Wearable Devices for Real-Time physiological monitoring. Department of Biomedical Engineering, Worcester Polytechnic Institute. [Online]. Available: http://www.wpi.edu/academics/Depts/BME/Research/mendelson.html

[9] T. Malina, A. Folkers, and U.G. Hoffman, "Real-time EEG processing based on wavelet transformation," in *12th Nordic Baltic Conference on Biomedical Engineering and Medical Physics*, June 2002, pp. 166-167.

[10] *CARDIC : Multiparametric Biomedical Front-End*, Aurelia Microelettronica Srl, Italy, 2009.

[11] J. Dong, S. Zhang, and X. Jia, "A portable intelligent ECG monitor based on wireless internet and embedded system technology" in *2008 International Conference on Biomedical Engineering and Informatics*, May 2008, pp. 553 - 556.

[12] Katrien Marent. (2007 Apr.). Wireless Platforms for Biomedical Monitoring, IMEC International, Belgium. [Online]. Available: http://www2.imec.be/imec_com/imec-reports-two-wireless-platforms-for-biomedical-monitoring.php

[13] A.K. Gupta, "Monitoring the injured brain in the intensive care unit," *Journal of Postgraduate Medicine*, vol. 48, no. 3, pp. 218-225, July-September 2002.

[14] B. Wallace, A. Wagner, E. Wagner, and J. McDeavitt, "A history and review of quantitative electroencephalography in traumatic brain injury," *Journal of Head Trauma Rehabilitation*, vol. 16, no. 2, pp. 165-190, 2001.

[15] J.A. Chambers, *EEG Signal Processing*. Sussex, England: John Wiley and Sons Ltd., 2007.

[16] E. Niedermeyer and L. Da Silva, *Electroencephalography: Basic Principles, clinical applications and related fields*, 5th ed. Philadelphia, PA: Lippincott Williams &Wilkins, 2005.

[17] S. Ahmad, G. Grindlinger, and S. Desjardins, "Noninvasive cerebral oximetry in patients with traumatic brain injury (TBI)," *Critical Care Medicine*, vol. 32, no. 12 (Suppl.), p. A104, 2004.

[18] C.M. Dunham, K. Ransom, C. McAuley, B. Gruber, D. Mangalat, and L. Flowers, "Severe brain injury ICU outcomes are associated with cranial-arterial pressure index and noninvasive bispectral index and transcranial oxygen saturation: A prospective, preliminary study," *Critical Care Medicine*, vol. 10, no. 6, pp. R159-R168, November 2006.

[19] L.M. Bateman, C.S. Li, and M. Seyal, "Ictal hypoxemia in localization related epilepsy: Analysis of incidence, severity and risk factors," *Brain*, vol. 131, pp. 3239-3245, December 2008.

[20] K. van der Hiele, A. A. Vein, R. H. Reijntjes, R. G. Westendorp, E. L. Bollen, M. A. van Buchem, J. G. van Dijk, and H. A. Middelkoop, "EEG correlates in the spectrum of cognitive decline," *Clinical Neurophysiology*, vol. 118, no. 9, 1931-1939, 2007.

[21] R. Thatcher, D. North, R. Curtin, R. Walker, C. Biver, J. Gomez, and A. Salazar, "An EEG severity index of traumatic brain injury," *Journal of Clinical Neuropsychiatry,* vol. 13, no. 1, pp. 77-87, Winter 2001.

[22] B. Wahlund, P. Piazza, D. von Rosen, B. Liberg, and H. Liljenstrom, "Seizure (Ictal) – EEG characteristics subgroup depressive disorder in patients receiving ECT – A preliminary study and multivariate approach," *Computational Intelligence and Neuroscience,* 2009, pp. 73-80.

[23] J. Meythaler and T. Novack. (2009, Feb.). Post traumatic seizures following head injury. *UAB Traumatic Brain Injury Model System*. [Online] Available: http://main.uab.edu/tbi/

[24] R.W. Thatcher, "Electroencephalography and mild traumatic brain injury," in *Foundations of Sport-Related Brain Injuries*. New York, NY: Springer, 2006, pp. 241-265.

[25] J. Richman and J. R. Moorman, "Physiological time-series analysis using approximate entropy and sample entropy," *American Journal of Physiology - Heart and Circulatory Physiology,* vol. 278, pp. H2039-H2049, June 2000.

[26] L. Fausett, *Fundamentals of Neural Network*, US ed. Upper Saddle River, NJ: Prentice Hall, 1993.

[27] EEG Time Series Database, Department of Epileptology, University of Bonn, Germany, May 2009. [Online]. Available: http://epileptologie-bonn.de/

[28] Multiple - Back Propagation ANN Training Software, N. M. Lopes, Instituto Politecnico da Guarda, October 2009. [Online]. Available: http://dit.ipg.pt/MBP/

[29] Leon3 SPARC Processor, Gaisler Research, November 2009. [Online]. Available: http://www.gaisler.com

[30] J. Gaisler, E. Catovic, M. Isomäki, K. Glembo, and S. Habinc, GRLIB IP core user's manual version 1.0.20, February 2009. [Online]. Available: http://gaisler.com/products/grlib/grip.pdf

[31] Advanced Microcontroller Bus Architecture, ARM Ltd., November 2009. [Online]. Available: http://www.arm.com/products/solutions/AMBAHomePage.html

[32] S.A. Kingsley, S. Sriram, A. Pollick and J. Marsh, "Revolutionary optical sensor for physiological monitoring in the battlefield," in *Proc. SPIE,* vol. 5403, September 2004, pp. 68-77.

APPENDIX A

RBPE VARIANCE MODULE CODE

In this section some snippets of C code are provided to understand how the various inputs and configuration parameters are provided to the RBPE variance module.

```
// structure of registers for the variance module

struct var_regs
{
volatile int result;        // result register
volatile int status;        // status register
volatile int control;       // control register
volatile int data;          // data register
volatile int scaler;        // scaler register
};

// function to read the value in a register
// assembly instruction lda of the SPARC ISA is used
// lda is the assembly instruction to load from alternate
// address space
static inline int loadmem(int addr)
{
  int tmp;
  asm volatile (" lda [%1]1, %0 ": "=r"(tmp):"r"(addr));
  return tmp;
}

// mapping the register to the address space within the
// leon3 system. Base address is 0x80000700

struct var_regs *var = (struct var_regs *)0x80000700;

// loading parameters into the control register
// sample size is 42. So SampleCount is 41
// sample size squared is 1764. So SampleCountSq is 1764
// scaler value is 10000
// clear the register before loading data

var->control = 0;
```

```
var->control = (1764 << 8) + (41);
// load the value of scaler
var->scaler = 10000;

// loading the samples sequentially

var->data = 100;            // sample value 100
var->data = 124;            // sample value 124

// polling the complete bit in the status register

while ((loadmem((int)&var->status) & 0x00000002)
== 0x00000002);

// reading out the result

variance = loadmem((int)&var->result);
```

APPENDIX B

RBPE NEURAL NETWORK MODULE CODE

In this section some snippets of code written in C are provided to help understand the process of loading the module configuration parameters.

```
//structure of registers in the neural network

struct ann_regs
{
volatile int weights[32];   // weight register
volatile int control;       // control register
volatile int input;         // input register
volatile int result;        // result register
volatile int status;        // status register
};

// function to read the value in a register
// assembly instruction lda of the SPARC ISA is used
// lda is the assembly instruction to load from alternate
// address space
static inline int loadmem(int addr)
{
  int tmp;
  asm volatile (" lda [%1]1, %0 ": "=r"(tmp):"r"(addr));
  return tmp;
}

// mapping the register to the address space within the
// leon3 system. Base address for the network module
// is 0x80000600

struct ann_regs *ann = (struct ann_regs *)0x80000600;

// function to load the weights into weight registers
int load_weights()
{
ann->weights[0] = 0x002d0000;    // for 1st input neuron
ann->weights[1] = 0x002b0000;    // for 2nd input neuron
ann->weights[2] = 0x002b0000;    // for 3rd input neuron
ann->weights[3] = 0x002c8000;    // for 4th input neuron
// next 5 weights are for 1st neuron in the hidden layer
```

```
ann->weights[4] = 0xfffd7126;
ann->weights[5] = 0xffffd931;
ann->weights[6] = 0xfffff038;
ann->weights[7] = 0xfffea73c;
ann->weights[8] = 0xfffef674;
// next 5 for 2nd neuron in the hidden layer
ann->weights[9] = 0x00061419;
ann->weights[10] = 0x00012f52;
ann->weights[11] = 0x0001bec6;
ann->weights[12] = 0x00015779;
ann->weights[13] = 0x0001dcc2;
// next 5 for 3rd neuron in the hidden layer
ann->weights[14] = 0xffd7d70d;
ann->weights[15] = 0xfff79050;
ann->weights[16] = 0xfff61535;
ann->weights[17] = 0xfff61247;
ann->weights[18] = 0xfff3abaf;
// we are using a 3 neuron hidden layer
// next 5 weights are zeros
ann->weights[19] = 0x00000000;
ann->weights[20] = 0x00000000;
ann->weights[21] = 0x00000000;
ann->weights[22] = 0x00000000;
ann->weights[23] = 0x00000000;
// next 5 weights are for output node
ann->weights[24] = 0x00025108;
ann->weights[25] = 0xffffff8c;
ann->weights[26] = 0x000054d0;
ann->weights[27] = 0xffef8beb;
ann->weights[28] = 0x00000000;
ann->weights[29] = 0x00000000;
ann->weights[30] = 0x00000000;
ann->weights[31] = 0x00000000;
}

// loading the input neuron's with input values
ann->input = 9;

// polling the computation complete bit
while ((loadmem((int)&ann->status) & 0x00000001) ==
0x00000001);

// reading the result
// bit 16 will be a 0 or 1. 0 for normal, 1 for seizure
result = loadmem((int)&reg) & 0x00010000) >> 16)
```

APPENDIX C

SAMPLE PROGRAM

A sample C program is provided to show how the variance and neural network modules are used together.

```
#include <stdio.h>

struct ann_regs
{
volatile int weights[32];
volatile int control;
volatile int input;
volatile int result;
volatile int status;
};

struct var_regs
{
volatile int result;
volatile int status;
volatile int control;
volatile int data;
volatile int scaler;
};

struct ann_regs *ann = (struct ann_regs *)0x80000600;
struct var_regs *var = (struct var_regs *)0x80000700;

int load_weights()
{
ann->weights[0] = 0x002d0000;
ann->weights[1] = 0x002b0000;
ann->weights[2] = 0x002b0000;
ann->weights[3] = 0x002c8000;
ann->weights[4] = 0xfffd7126;
ann->weights[5] = 0xffffd931;
ann->weights[6] = 0xfffff038;
ann->weights[7] = 0xfffea73c;
ann->weights[8] = 0xfffef674;
ann->weights[9] = 0x00061419;
```

```
ann->weights[10] = 0x00012f52;
ann->weights[11] = 0x0001bec6;
ann->weights[12] = 0x00015779;
ann->weights[13] = 0x0001dcc2;
ann->weights[14] = 0xffd7d70d;
ann->weights[15] = 0xfff79050;
ann->weights[16] = 0xfff61535;
ann->weights[17] = 0xfff61247;
ann->weights[18] = 0xfff3abaf;
ann->weights[19] = 0x00000000;
ann->weights[20] = 0x00000000;
ann->weights[21] = 0x00000000;
ann->weights[22] = 0x00000000;
ann->weights[23] = 0x00000000;
ann->weights[24] = 0x00025108;
ann->weights[25] = 0xffffff8c;
ann->weights[26] = 0x000054d0;
ann->weights[27] = 0xffef8beb;
ann->weights[28] = 0x00000000;
ann->weights[29] = 0x00000000;
ann->weights[30] = 0x00000000;
ann->weights[31] = 0x00000000;
return 0;
}

int load_input(int ineuron)
{
ann->input = ineuron;
}


static inline int loadmem(int addr)
{
  int tmp;
  asm volatile (" lda [%1]1, %0 "
      : "=r"(tmp)
      : "r"(addr)
    );
  return tmp;
}


int var_test(int addr)
{
volatile int ineuron = 0;
```

```
// loading the control register
// SampleCount - 41
// SampleCountSq bits - 1764
// Value for Scaler - 10000
var->scaler = 10000;
var->control = (1764 << 8) + (41);

// now load the neural network weight registers
load_weights();

// send sample set values to variance module
var->data = 100;
var->data = 124;
var->data = 153;
var->data = 185;
var->data = 210;
var->data = 220;
var->data = 216;
var->data = 222;
var->data = 240;
var->data = 265;
var->data = 298;
var->data = 330;
var->data = 362;
var->data = 381;
var->data = 391;
var->data = 390;
var->data = 390;
var->data = 392;
var->data = 395;
var->data = 397;
var->data = 397;
var->data = 393;
var->data = 380;
var->data = 356;
var->data = 334;
var->data = 322;
var->data = 320;
var->data = 322;
var->data = 338;
var->data = 332;
var->data = 317;
var->data = 263;
var->data = 195;
var->data = 142;
```

```
var->data = 108;
var->data = 97;
var->data = 80;
var->data = 80;
var->data = 85;
var->data = 39;
var->data = -139;
var->data = -816;

// poll the computation complete bit
while ((loadmem((int)&var->status) & 0x00000002) ==
0x00000002);

// read the result
ineuron = loadmem((int)&var->result);
// send to the input neuron
load_input(ineuron);

// repeat the same procedure for other samples
var->data = -1202
......
......
ineuron = loadmem((int)&var->result);
load_input(ineuron);
......
......
ineuron = loadmem((int)&var->result);
load_input(ineuron);
.......
.......
ineuron = loadmem((int)&var->result);
load_input(ineuron);
// we have loaded the 4 input neurons
// computation has started
// poll the complete bit
while ((loadmem((int)&ann->status) & 0x00000001) ==
0x00000001);

// check the result for a seizure
if ((loadmem((int)&reg) & 0x00010000) >> 16) == 1)
{
  printf("Seizure detected");
}
return 0;
}
```